



Sun™ Grid Engine 5.2.3 Manual

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 U.S.A.
650-960-1300

Part No. 816-2077-10
July 2001

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303-4900 U.S.A. All rights reserved.

This product or document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, AnswerBook2, docs.sun.com, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303-4900 Etats-Unis. Tous droits réservés.

Ce produit ou document est distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, AnswerBook2, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

1. Quick Start Guide 15

Introduction 15

Document Organization 15

Sun Grid Engine Components and Concepts 16

How Sun Grid Engine Operates 16

A “Sun Grid Engine-Bank” 16

Jobs and Queues - the Sun Grid Engine World 17

Sun Grid Engine Components 17

Hosts 18

Daemons 18

Queues 19

Client Commands 19

Quick Start Installation Guide 22

Prerequisites 22

Installation Accounts 22

Creating the Installation Directory 22

Adding a Service to the Services Database 23

Reading the Distribution Media 23

Installing a Default Sun Grid Engine System for your Cluster 24

| | |
|--|-----------|
| Installing the Master Host | 24 |
| The Execution Host Installation | 25 |
| The Default System Configuration | 26 |
| Quick Start User's Guide | 28 |
| Running a Simple Job | 28 |
| Basic Use of the Graphical User's Interface qmon | 30 |
| A Guide Through the Sun Grid Engine Manual Set | 33 |
| The <i>Sun Grid Engine Installation and Administration Guide</i> | 33 |
| The <i>Sun Grid Engine User's Guide</i> | 35 |
| The <i>Sun Grid Engine Reference Manual</i> | 37 |
| Glossary of Sun Grid Engine Terms | 37 |
| 2. Installation and Administration Guide | 41 |
| Introduction | 41 |
| Installation | 42 |
| Overview | 42 |
| Phase 1 - Planning | 42 |
| Phase 2 - Install the Software | 43 |
| Phase 3 - Verify the Installation | 43 |
| Planning | 43 |
| Prerequisite Tasks | 43 |
| Installation Plan | 49 |
| Reading the Distribution Media | 49 |
| Installing the Master Host | 50 |
| Installing Execution Hosts | 51 |
| Installing Administration and Submit Hosts | 52 |
| Verifying the Installation | 52 |
| Architectural Dependencies | 55 |

| | |
|--|----|
| Master and Shadow Master Configuration | 55 |
| Sun Grid Engine Daemons and Hosts | 56 |
| Classification | 56 |
| Configuring Hosts | 57 |
| Administrative Hosts | 58 |
| Submit Hosts | 60 |
| Execution Hosts | 62 |
| Killing and Restarting Daemons | 69 |
| Cluster Configuration | 70 |
| The Basic Cluster Configuration | 70 |
| Displaying the Basic Cluster Configurations | 71 |
| Modifying the Basic Cluster Configurations | 71 |
| Displaying the Cluster Configuration with qmon | 72 |
| Modifying global and Host Configurations with qmon | 73 |
| Configuring Queues | 75 |
| Configuring Queues with qmon | 75 |
| Configuring General Parameters | 76 |
| Configuring Execution Method Parameters | 77 |
| Configuring Checkpointing Parameters | 78 |
| Configuring Load and Suspend Thresholds | 79 |
| Configuring Limits | 81 |
| Configuring User Complexes | 82 |
| Configuring Subordinate Queues | 84 |
| Configuring User Access | 85 |
| Configuring Owners | 86 |
| Configuring Queues from the Command-line | 87 |
| The Complexes Concept | 88 |

| | |
|--|-----|
| Complex Types | 90 |
| The <i>Queue</i> Complex | 90 |
| The <i>Host</i> Complex | 91 |
| The <i>Global</i> Complex: | 93 |
| <i>User Defined</i> Complexes | 93 |
| Consumable Resources | 96 |
| Setting Up Consumable Resources | 97 |
| Examples | 99 |
| Configuring Complexes | 108 |
| Queue Calendars | 109 |
| Configuration with <code>qmon</code> | 109 |
| Command-line Configuration | 112 |
| Load Parameters | 113 |
| The Default Load Parameters | 113 |
| Adding Site Specific Load Parameters | 114 |
| How to Write Your Own Load Sensors | 114 |
| Managing User Access | 117 |
| Manager Accounts | 118 |
| Configure Manager Accounts with <code>qmon</code> | 118 |
| Configure Manager Accounts from the Command-line | 119 |
| Operator Accounts | 120 |
| Configure Operator Accounts with <code>qmon</code> | 120 |
| Configure Operator Accounts from the Command-line | 121 |
| Queue Owner Accounts | 121 |
| User Access Permissions | 122 |
| Configure User Access Lists with <code>qmon</code> | 122 |
| Configure User Access from the Command-line | 124 |
| Scheduling | 124 |

| | |
|--|-----|
| Overview | 124 |
| Scheduling Strategies | 125 |
| What Happens in a Scheduler Interval | 126 |
| Scheduler Monitoring | 126 |
| Scheduler Configuration | 127 |
| Default Scheduling | 127 |
| Scheduling Alternatives | 127 |
| Changing the Scheduler Configuration via qmon | 131 |
| The Sun Grid Engine Path Aliasing Facility | 134 |
| Configuring Default Requests | 135 |
| Setting Up a Sun Grid Engine User | 137 |
| Customizing qmon | 138 |
| Gathering Accounting and Utilization Statistics | 139 |
| Checkpointing Support | 140 |
| Checkpointing Environments | 141 |
| Configuring Checkpointing Environments with qmon | 141 |
| Command-line Configuration of Checkpointing Environment. | 144 |
| Support of Parallel Environments | 145 |
| Parallel Environments | 145 |
| Configuring PEs with qmon | 145 |
| Configuring PEs from the Command-line | 149 |
| The PE Start-up Procedure | 150 |
| Termination of the PE | 151 |
| Tight Integration of PEs and Sun Grid Engine | 152 |
| The Sun Grid Engine Queuing System Interface (QSI) | 153 |
| Motivation | 153 |
| How Jobs for Another Queueing System are Processed | 153 |
| The QSI Configuration File | 154 |

| | |
|--|-----|
| Setting Up QS Command Procedures | 155 |
| An Example of a QSI file | 156 |
| Monitoring QSI Daemons and Jobs | 157 |
| Trouble Shooting | 158 |
| Scheduler Monitoring | 158 |
| Retrieving Error Reports | 158 |
| Running Sun Grid Engine Programs in Debug Mode | 159 |

3. User's Guide 161

| | |
|---|-----|
| Introduction | 161 |
| Sun Grid Engine User Types and Operations | 162 |
| Navigating through the Sun Grid Engine System | 163 |
| Overview on Host Functionality | 163 |
| The Master Host | 164 |
| Execution Hosts | 164 |
| Administration Hosts | 164 |
| Submit Hosts | 165 |
| Queues and Queue Properties | 165 |
| The Queue Control qmon Dialogue | 165 |
| Show Properties with the qmon Object Browser | 165 |
| Queue Information from the Command-line | 166 |
| Requestable Attributes | 168 |
| User Access Permissions | 171 |
| Managers, Operators and Owners | 173 |
| Submit Batch Jobs | 173 |
| Shell Scripts | 173 |
| Example Script File | 174 |
| Submitting Sun Grid Engine Jobs | 175 |

| | |
|--|-----|
| Submitting jobs with qmon (Simple Example) | 175 |
| Submitting jobs with qmon (Extended Example) | 177 |
| Submitting Jobs with qmon (Advanced Example) | 181 |
| Extensions to Regular Shell Scripts | 185 |
| Submitting Jobs from the Command-line | 189 |
| Default Requests | 190 |
| Resource Requirement Definition | 191 |
| How Sun Grid Engine Allocates Resources | 194 |
| Array Jobs | 194 |
| Parallel Jobs | 196 |
| Submitting Jobs to Other Queueing Systems | 198 |
| How Sun Grid Engine Jobs Are Scheduled | 199 |
| Job Scheduling | 199 |
| Queue Selection | 200 |
| Submit Interactive Jobs | 200 |
| Submit Interactive Jobs with qmon | 201 |
| Submitting Interactive Jobs with qsh | 203 |
| Submitting Interactive Jobs with qlogin | 204 |
| Transparent Remote Execution | 204 |
| Remote Execution with qrsh | 204 |
| Qrsh Usage | 205 |
| Transparent Job Distribution with qtcsh | 206 |
| Qtcsh Usage | 206 |
| Parallel Makefile Processing with qmake | 208 |
| Qmake Usage | 209 |
| Checkpointing Jobs | 211 |
| User Level Checkpointing | 211 |
| Kernel Level Checkpointing | 211 |

| | |
|--|-----|
| Migration of Checkpointing Jobs | 212 |
| Composing a Checkpointing Job Script | 212 |
| Submit/Monitor/Delete a Checkpointing Job | 213 |
| Submit a Checkpointing Job with qmon | 214 |
| File System Requirements | 215 |
| Monitoring and Controlling Sun Grid Engine Jobs | 216 |
| Monitoring and Controlling Jobs with qmon | 216 |
| Additional Information with the qmon Object Browser | 226 |
| Monitoring with qstat | 227 |
| Monitoring by Electronic Mail | 230 |
| Controlling Sun Grid Engine Jobs from the Command-line | 230 |
| Job Dependencies | 231 |
| Controlling Queues | 232 |
| Controlling Queues with qmon | 232 |
| Controlling Queues with qmod | 236 |
| Customizing qmon | 237 |

4. Reference Manual 239

| | |
|-------------------------|-----|
| Introduction | 239 |
| Typographic Conventions | 239 |
| SGE_INTRO(1) | 240 |
| SGE_CKPT(1) | 243 |
| QACCT(1) | 245 |
| QCONF(1) | 249 |
| QDEL(1) | 264 |
| QHOLD(1) | 267 |
| QHOST(1) | 270 |
| QMAKE(1) | 275 |

QMOD(1) 279
QMON(1) 282
QRLS(1) 291
QSELECT(1) 294
QSTAT(1) 297
QTCSH(1) 306
SUBMIT(1) 310
ACCESS_LIST(5) 332
ACCOUNTING(5) 333
CALENDAR_CONF(5) 336
CHECKPOINT(5) 340
COD_REQUEST(5) 344
CODINE_ALIASES(5) 346
SGE_CONF(5) 348
SGE_H_ALIASES(5) 366
SGE_PE(5) 367
COMPLEX(5) 372
HOST_CONF(5) 379
PROJECT(5) 383
QSI_CONF(5) 385
QTASK(5) 391
QUEUE_CONF(5) 393
SCHED_CONF(5) 407
SHARE_TREE(5) 414
USER(5) 415
COD_COMMD(8) 417
COD_EXECD(8) 420
COD_QMASTER(8) 423

COD_QSTD(8) 426
COD_SCHEDD(8) 429
COD_SHADOWD(8) 431
COD_SHEPHERD(8) 433
CODCOMMDCTL(8) 435

Preface

How This Book Is Organized

Chapter 1 gives an overview on the Sun™ Grid Engine system, its features and components. The *Sun Grid Engine Quick Start Guide* also contains a quick installation procedure for a small sample Sun Grid Engine configuration and a glossary of terms commonly used in the Sun Grid Engine manual set.

Chapter 2 is provided for those responsible for the cluster administration. See the *Sun Grid Engine Installation and Administration Guide* for a description of the Sun Grid Engine cluster management facilities.

Chapter 3 is an introduction for the user to the Sun Grid Engine.

Chapter 4 is a reference manual for a detailed discussion of all available Sun Grid Engine commands.

Typographic Conventions

| Typeface | Meaning | Examples |
|------------------|--|---|
| AaBbCc123 | The names of commands, files, and directories; on-screen computer output | Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail. |
| AaBbCc123 | What you type, when contrasted with on-screen computer output | % su Password: |
| <i>AaBbCc123</i> | Book titles, new words or terms, words to be emphasized | Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this. |
| | Command-line variable; replace with a real name or value | To delete a file, type <code>rm filename</code> . |

Ordering Sun Documentation

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at:

<http://www.fatbrain.com/documentation/sun>

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can email your comments to Sun at:

docfeedback@sun.com

Please include the part number (816-2077-10) of your document in the subject line of your email.

Quick Start Guide

Introduction

Sun Grid Engine (Computing in Distributed Networked Environments) is a *load management* tool for heterogeneous, distributed computing environments. Sun Grid Engine provides an effective method for distributing the batch workload among multiple computational servers. In doing so, it increases the productivity of all of the machines and simultaneously increases the number of jobs that can be completed in a given time period. Also, by increasing the productivity of the workstations, the need for outside computational resources is reduced.

Document Organization

The subsequent sections in this document will focus on the following goals:

- Sun Grid Engine Components and Concepts
 - explains Sun Grid Engine's key concepts and its major components. This chapter and the "Glossary of Sun Grid Engine Terms" provide the background for using Sun Grid Engine.
- Quick Start Installation Guide
 - explains how to install a minimal Sun Grid Engine configuration that will enable you to run your first example jobs. This minimal set-up does not represent the full Sun Grid Engine functionality. However, the quick start configuration may be extended later by means of the Sun Grid Engine administration toolset which is described in detail in the *Sun Grid Engine Installation and Administration Guide*.

- Quick Start User's Guide

introduces to the usage (job submission, monitoring) of the basic Sun Grid Engine system installed following the procedures in section "Quick Start Installation Guide".

- A Guide Through the Sun Grid Engine Manual Set

provides an overview and short description of the contents of the Sun Grid Engine manual set consisting of the *Sun Grid Engine Installation and Administration Guide*, the *Sun Grid Engine User's Guide* and the *Sun Grid Engine Reference Manual*.

- Glossary of Sun Grid Engine Terms

gives a definition of commonly used terms in the context of Sun Grid Engine and resource management in general.

Sun Grid Engine Components and Concepts

How Sun Grid Engine Operates

Sun Grid Engine accepts jobs from the outside world, puts them in a holding area until they can be executed, sends them from the holding area to an execution device, manages them during execution and logs the record of their execution when they are finished.

Let's use the analogy of a counter-room in a bank, of counters and of customers to become familiar with the Sun Grid Engine world.

A "Sun Grid Engine-Bank"

Say that customers are waiting in the counter-room of a bank to be served. Each customer has different requirements. A customer might want to retrieve money from an account while another customer is seeking investment consulting and has an appointment. There may be many counters providing the sought service for one customer, but only a single counter being suitable for another customer. Sun Grid Engine would organize the service in a counter-room slightly different than you may be used to from your own bank:

- When entering the counter-room customers have to declare their name, their affiliations (such as representing a company) and their requirements. In addition, the time when they entered will be denoted.

- Whenever a counter becomes available, this information is used to select among the waiting customers those, for which the counter is suitable and finally to dispatch the customer to the counter who has the highest priority or who waited to be serviced for the longest time.
- In a “Sun Grid Engine-bank” a counter may be able to provide service to several customers at the same time. Sun Grid Engine will try to assign new customers to the “least loaded” and suitable counter.

Jobs and Queues - the Sun Grid Engine World

In a Sun Grid Engine system, jobs correspond to bank customers, jobs wait in a holding area instead of a counter-room and queues located on computational servers provide services for jobs as opposed to customers being served at counters. Like in the case of bank customers, the requirements of the jobs may be very different and only certain queues may be able to provide the corresponding service, but the requirements typically consist of available memory, execution speed, available software licenses and similar needs.

Corresponding to our analogy, Sun Grid Engine arbitrates available resources and job requirements in the following fashion:

- A user who submits a job to Sun Grid Engine declares a requirement profile for the job. In addition, the identity of the user and its affiliation with projects or user groups is retrieved. The time of submission is also stored.
- As soon as a queue becomes available for execution of a new job, Sun Grid Engine determines suitable jobs for the queue and will dispatch the job with the highest priority or longest waiting time.
- Sun Grid Engine queues may allow execution of many jobs concurrently at the same time. Sun Grid Engine will try to start new jobs in the least loaded and suitable queue.

Sun Grid Engine Components

Figure 1-1 on page 21 displays the most important Sun Grid Engine components and their interaction in the system. A short explanation of the components is given in the following subsections.

Hosts

- Master Host:

The master host is central for the overall cluster activity. It runs the master daemon `cod_qmaster` and the scheduler daemon `cod_schedd`. Both daemons control all Sun Grid Engine components such as queues and jobs and maintain tables about the status of the components, about user access permissions and the like.

- Execution Host:

Execution hosts are nodes having permission to execute Sun Grid Engine jobs. Therefore, they are hosting Sun Grid Engine queues and run the Sun Grid Engine execution daemon `cod_execd`.

- Administration Host:

Permission can be given to hosts to carry out any kind of administrative activity for Sun Grid Engine.

- Submit Host:

Submit hosts allow for submitting and controlling batch jobs only. In particular a user being logged into a submit host can submit jobs via `qsub`, can control the job status via `qstat` or run Sun Grid Engine's OSF/1 Motif graphical user's interface `qmon`.

Note – A host may belong to more than one of the above described classes.

Note – The master host is an administrative and submit host by default.

Daemons

- Master Daemon:

The master daemon `cod_qmaster`. The center of the cluster's management and scheduling activities. `cod_qmaster` maintains tables about hosts, queues, jobs, system load and user permissions. It receives scheduling decisions from `cod_schedd` and requests actions from `cod_execd` on the appropriate execution hosts.

- Scheduler Daemon:

The scheduling daemon `cod_schedd`. It maintains an up-to-date view of the cluster's status with the help of `cod_qmaster`. It makes scheduling decisions:

- what jobs are dispatched to which queues.

It forwards these decisions to `cod_qmaster` which initiates the actions decided on.

- Execution Daemon:

The execution daemon `cod_execd`. It is responsible for the queues on its host and for the execution of jobs in these queues. Periodically it forwards information such as job status or load on its host to `cod_qmaster`.

- Communication Daemon:

The communication `cod_commd`. It communicates over a well-known TCP port. It is used for all communication among Sun Grid Engine components.

Queues

A Sun Grid Engine queue is a container for a class of jobs allowed to execute on a particular host concurrently. A queue determines certain job attributes; for example, whether it may be migrated or not. Throughout their lifetimes, running jobs are associated with their queue. Association with a queue affects some of the things that can happen to a job. For example, if a queue is suspended, all the jobs associated with that queue are also suspended.

In Sun Grid Engine there is no need to submit jobs directly to a queue. You only need to specify the requirement profile of the job (e.g., memory, operating system, available software, etc.) and Sun Grid Engine will dispatch the job to a suitable queue on a low loaded host automatically. If a job is submitted to a particular queue, the job will be bound to this queue and to its host, and thus Sun Grid Engine will be unable to select a lower loaded or better suited device.

Client Commands

Sun Grid Engine's command line user interface is a set of ancillary programs (commands) that let you manage queues, submit and delete jobs, check job status and suspend/enable queues and jobs. Sun Grid Engine encompasses the following set of ancillary programs:

- `qacct`:

extracts arbitrary accounting information from the cluster logfile.

- `qalter`:

changes the attributes of already submitted but still pending jobs.

- `qconf`:

provides the user interface for cluster and queue configuration.

- `qdel`:

provides the means for a user/operator/manager to send signals to jobs or subsets thereof.

- `qhold`:
holds back submitted jobs from execution.
- `ghost`:
displays status information about Sun Grid Engine execution hosts.
- `qlogin`:
initiates a telnet or similar login session with automatic selection of a low loaded and suitable host.
- `qmake`:
is a replacement for the standard Unix make facility. It extends make by its ability to distribute independent make steps across a cluster of suitable machines.
- `qmod`:
allows the *owner* to suspend or enable a queue (all currently active processes associated with this queue are also signaled).
- `qmon`:
provides an X-windows Motif command interface and monitoring facility.
- `qresub`:
creates new jobs by copying currently running or pending jobs.
- `grls`:
releases jobs from holds previously assigned to them e.g. via `qhold` (see above).
- `qrsh`:
can be used for various purposes such as providing remote execution of interactive applications via Sun Grid Engine comparable to the standard Unix facility `rsh`, to allow for the submission of batch jobs which, upon execution, support terminal I/O (standard/error output and standard input) and terminal control, to provide a batch job submission client which remains active until the job has finished or to allow for the Sun Grid Engine-controlled remote execution of the tasks of parallel jobs.
- `qselect`:
prints a list of queue names corresponding to specified selection criteria. The output of `qselect` is usually fed into other Sun Grid Engine commands to apply actions on a selected set of queues.
- `qsh`:
opens an interactive shell (in an *xterm*) on a low loaded host. Any kind of interactive jobs can be run in this shell.
- `qstat`:
provides a status listing of all jobs and queues associated with the cluster.

- **qsub:**
is the user interface for submitting a job to Sun Grid Engine.
- **qtcsch:**
is a fully compatible replacement for the widely known and used Unix C-Shell (**csh**) derivative **tcsh**. It provides a command-shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts via Sun Grid Engine.

All programs communicate with **cod_qmaster** via **cod_commd**. This leads to the schematic view of the component interaction in Sun Grid Engine shown in figure 1-1 on page 21

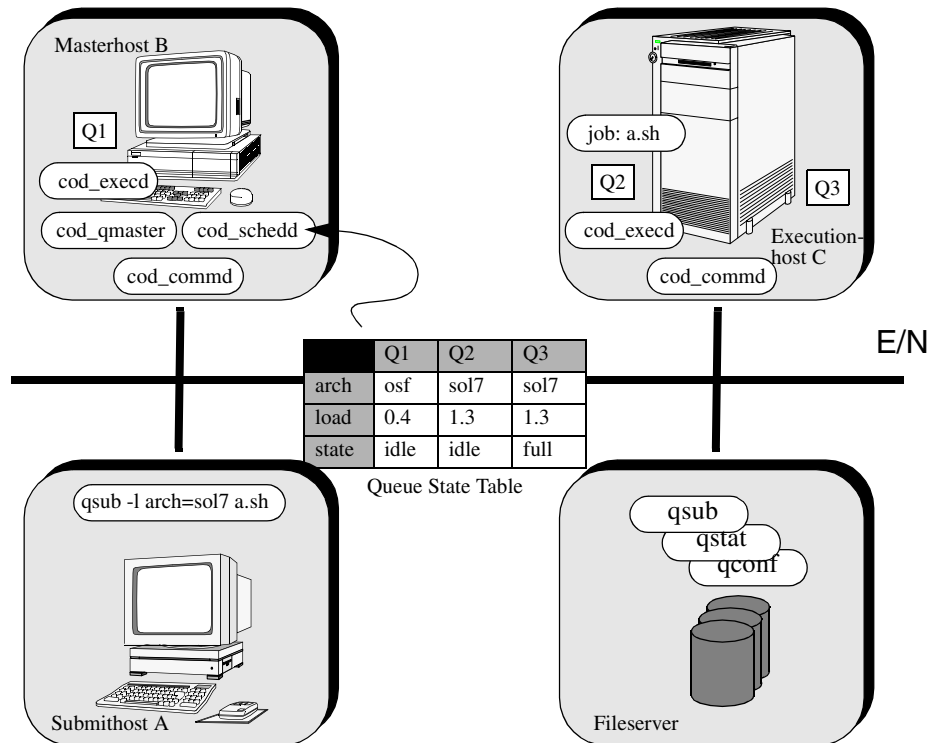


FIGURE 1-1 Component Interaction in Sun Grid Engine

Quick Start Installation Guide

Note – In the following the conditions for applicability of the *quick start* installation procedures are described. If your environment does not permit any of the prerequisites outlined below, the quick installation procedure cannot be used. In this case, please refer to the *Sun Grid Engine Installation and Administration Guide* for detailed information on how to install Sun Grid Engine under more restricted conditions.

Prerequisites

Installation Accounts

An *Administrator* account should be created. The Administrator can be an existing administrative login or a new login such as *codadmin*. This account will own all of the files in the Sun Grid Engine installation and spooling directories and it can be used to configure and administer the cluster once it is installed. This user should not be *root*. This account must exist prior to installation!

If you intend to use *root* for file ownership, the user *root* must have full write permissions on all hosts in the directory where Sun Grid Engine is installed. Usually a shared (NFS) filesystem is not exported for the user *root* to allow write permission.

Creating the Installation Directory

In preparation of using the Administrator account, you can create the installation directory preferably on a network-wide share file system using the following sequence of commands:

```
% mkdir -p <install_dir>
% chown <adminuser> <install_dir>
% chmod 755 <install_dir>
```

The directory created by this or a similar procedure will be referred to as Sun Grid Engine *root directory* for the remainder of this manual.

Adding a Service to the Services Database

Sun Grid Engine uses a TCP port for communication. All hosts in the cluster must use the same port number. The port number can be placed in several places. For example:

- NIS (Yellow Pages) services or NIS+ database. Add the following to the services database:

CODE EXAMPLE 1-1 communication port for Sun Grid Engine

```
% cod_commd 535/tcp
```

- `/etc/services` on each machine. If NIS is not running at your site, then the above services can be added to the `/etc/services` file on each machine.

It is recommended to use a privileged port below 600 to avoid conflicts with applications which bind ports near below 1024 or ports higher than 1024 dynamically.

Reading the Distribution Media

Sun Grid Engine is distributed either on CD-ROM or as archive file through Internet download. The distribution will consist of a *tape archive* (`tar`) directly written on the medium.

To unpack the Sun Grid Engine distribution, please login as the account you selected for the installation (see section “Prerequisites” on page 22) to the host from which you plan to read in the Sun Grid Engine distribution media and change your working directory to the Sun Grid Engine root directory. Then read in the distribution media with the following command:

```
% cd codine_root_dir  
% tar -xvpf distribution_source
```

where *codine_root_dir* is the pathname of the Sun Grid Engine root directory and *distribution_source* is either the name of the tape archive file on hard disk or CD-ROM. This will read in the Sun Grid Engine installation kit.

Installing a Default Sun Grid Engine System for your Cluster

A default Sun Grid Engine system consists of a so called *master host* and an arbitrary number of *execution hosts*. The master host controls the overall cluster activity while the execution hosts control the execution of the jobs being assigned to them by the master host. A single host may concurrently act as a master host and as an execution host.

Note – Please install the master host first and then conclude with the installation of the execution hosts in arbitrary sequence.

Installing the Master Host

Select a machine as the master host. It should fulfill the following characteristics:

- The selected system should not be overloaded with other tasks.
- The master host should provide for enough available main memory to run the necessary Sun Grid Engine daemons.

Note – The required amount strictly depends on the size of your cluster and the number of jobs in the system to be expected. For clusters up to a few dozen hosts and in the order of 100 jobs 10 MB of free memory should be sufficient.

Note – For very large clusters (in the order or above 1000 hosts and several 10000 jobs) you may well need 1 GB of memory.

Now, login to the selected machine. For an installation featuring all capabilities, you will need to install using the *root* account (files still may be owned by the Administrator account created in Section , “Prerequisites” on page 1-22. For a test installation you may also install as the Administrator user, but then only Administrator will be able to run jobs and Sun Grid Engine will have restricted capabilities with respect to monitoring system load and system control.

After logging in, please change directory to the Sun Grid Engine root directory. Then execute the master host installation procedure with the command:

```
% ./install_qmaster
```


If errors occur, the installation procedure will print a description of the error condition and you will have to check with the *Sun Grid Engine Installation and Administration Guide* to resolve the error.

During the installation you will be asked for a list of hosts you initially want to install. You should provide a list with all such hosts, since these hosts will be added as submit hosts and administrative hosts.

The installation of the execution hosts (see below) requires that all hosts are administrative hosts. If you plan to install Sun Grid Engine on many hosts, the installation script will give you the possibility to provide the path to a file which contains the list of all host names with one host per line.

The installation procedure requires some additional information. Most questions will provide useful defaults, which can simply be confirmed by pressing **<return>**.

The Execution Host Installation

As with the master host installation, the execution hosts should be installed using the *root* account to have access to all Sun Grid Engine facilities. Installing as *root* still allows that all files are owned by the Administrator account created in Section , “Prerequisites” on page 1-22. Installation using the Administrator account is only useful for test purposes and prohibits other users than Administrator to run jobs and does not allow Sun Grid Engine to provide full system monitoring and control capabilities.

Login as the account selected for the installation to one of the execution hosts specified during the master host installation procedure and go into the Sun Grid Engine root directory. Now execute the execution host installation procedure:

```
% ./install_execd
```

Again any errors are indicated by the installation procedure and require manual resolving by the help of the detailed information in the *Sun Grid Engine Installation and Administration Guide*.

The installation procedure will ask you whether default queues should be configured for your host. The queues defined in this case will have the characteristics described in section “The Default System Configuration” below.

If the procedure notifies you of successful completion of the execution host installation procedure, you can proceed likewise with the next execution host being on the list you entered during the master host installation. As soon as you are through with the list, your default Sun Grid Engine system is configured on your cluster and is ready to be used.

The following chapters provide you with an overview on the default configuration which has been installed and they guide you through the first steps of using Sun Grid Engine.

The Default System Configuration

Note – The following is a description of the Sun Grid Engine system as configured in your environment by the quick installation procedure. It is a minimal setup for testing purposes and may be changed or extended later-on at any time.

After successful completion of the master and execution host installations, the following basic Sun Grid Engine system has been configured on your cluster:

- Master Host:

The host on which you ran the master host installation procedure is configured to be the master host of your cluster. No shadow master hosts are configured to take over the master host's tasks in case of failure.

- Execution Hosts:

During the master host installation you are asked for a list of machines on which you want to install the Sun Grid Engine execution agent. During installation of these execution hosts, you can allow the installation procedure to create *queues* automatically on these hosts. A queue describes the profile (a list of attributes and requirements) of the jobs that can be run on that particular host. The queues being configured for the execution hosts by default show the following important characteristics:

- Queue name: *<unqualifiedhostname>.q*
- Slots (concurrent jobs): *<number_of_processors>*
- The queues provide unlimited system resources (such as memory, CPU-time etc.) to the jobs.
- The queues do not enforce any access restrictions for particular users or user groups. Any user with a valid account can run jobs in the queues.
- A load threshold of 1.75 per CPU will be configured (i.e., 1.75 processes attempting on average to get access to each CPU).

Note – The queue configurations likewise any other Sun Grid Engine configuration can be changed on-the-fly at any later stage while the system is in operation.

Note – If you invoked the execution host installation procedure on the master host also, the master host acts both as master and as execution host.

- **Administrative Accounts and Hosts:**

The master host and all execution hosts are configured to be allowed to execute administrative Sun Grid Engine commands. The only users that are allowed to administer Sun Grid Engine are the user *root* and the Administrator account described in section “Prerequisites”. If an unprivileged user installs Sun Grid Engine he is added to the list of Sun Grid Engine administrators too.

- **Submit Accounts and Hosts:**

If you installed under the root account any user with a valid account can submit and control Sun Grid Engine jobs. The user under which you installed Sun Grid Engine will be the only user to whom access is permitted otherwise (“Prerequisites” on page 22). The tasks of submitting jobs, controlling the Sun Grid Engine system activity or deleting jobs can be executed from either the master host or from any execution host.

- **Daemons:**

The following daemons are started up during system installation on the different hosts or may be invoked during normal system operation respectively:

- *cod_qmaster* runs on the master host only. It is the central cluster activity control daemon.
- *cod_schedd* is also invoked on the master host only. This daemon is responsible for distributing the workload in the Sun Grid Engine cluster.
- *cod_execd* is responsible for executing the jobs on an execution host and, therefore, is running on all execution hosts.
- One instance of *cod_shepherd* is run for each job being actually executed on a host. *cod_shepherd* controls the jobs process hierarchy and collects accounting data after the job has finished.
- *cod_commd* runs on each execution host and on the master host. The network of all *cod_commds* forms the network communication backbone of the Sun Grid Engine cluster.

Quick Start User's Guide

Running a Simple Job

Note – If the Sun Grid Engine system was installed as *root* with the quick installation procedure (described in “Quick Start Installation Guide”) any user account being valid on all machines of the Sun Grid Engine cluster can be used for the following tests. If Sun Grid Engine was installed under an unprivileged account you must login as that particular user to be able to run jobs (see “Prerequisites” for details).

Prior to executing any Sun Grid Engine command, you first need to set your executable search path and other environmental conditions properly. The easiest way to achieve the appropriate settings is to execute the command:

```
% source codine_root_dir/default/common/settings.csh
```

if one of *csh* or *tcsh* is the command interpreter you are using and *codine_root_dir* specifies the location of the Sun Grid Engine root directory selected at the beginning of the quick installation procedure. Alternatively you can execute:

```
# . codine_root_dir/default/common/settings.sh
```

if *sh*, *ksh* or *bash* is the command interpreter in use.

Note – You can add the above commands into your *.login*, *.cshrc* or *.profile* files (whichever is appropriate) to guarantee proper Sun Grid Engine settings for all interactive session you will start later-on.

Now you can try to submit the following simple job script to your Sun Grid Engine cluster (the job can be found in the file `examples/jobs/simple.sh` in your Sun Grid Engine root directory):

```
#!/bin/sh
#This is a simple example of a Sun Grid Engine batch script
#
# Print date and time
date
# Sleep for 20 seconds
sleep 20
# Print date and time again
date
# End of script file
```

The Sun Grid Engine command to submit such job scripts is:

```
% qsub simple.sh
```

if *simple.sh* is the name of the script file in which the above script is stored and if the file is located in your current working directory. The `qsub` command should confirm the successful job submission as follows:

```
your job 1 ("simple.sh") has been submitted
```

Now you can retrieve status information on your job via the command:

```
% qstat
```

You should receive a status report containing information about all jobs currently known to the Sun Grid Engine system and for each of them the so called *job ID* (the unique number being included in the submit confirmation), the name of the job script, the owner of the job, a state information (“r” means running), the submit or start time and eventually the name of the queue in which the job executes.

If no output is produced by the `qstat` command, no jobs are actually known to the system - for example, your job may already have finished. You can control the output of the finished jobs by checking their *stdout* and *stderr* redirection files. By default, these files are generated in the job owner’s home directory on the host which has executed the job. The names of the files are composed of the job script file name, an appended dot sign followed by an “o” for the *stdout* file and an “e” for the

stderr file and finally the unique job ID. Thus the stdout and stderr files of your first job can be found under the names `simple.sh.o1` and `simple.sh.e1` respectively.

Basic Use of the Graphical User's Interface qmon

A more convenient method of submitting and controlling Sun Grid Engine jobs and of getting an overview on the Sun Grid Engine system is the X-windows OSF/Motif graphical user's interface `qmon`. Among other facilities `qmon` provides a job submission menu and a job control dialog for the tasks of submitting and monitoring jobs.

`qmon` is simply invoked by typing:

```
% qmon
```

from the command line prompt. During start-up a message window is displayed. Afterwards the `qmon` main control panel will appear.



Click here and here

FIGURE 1-2 qmon main control menu

Now use the left mouse button to click on the Job Control and the Submit buttons thus opening the Job Control and the Submit dialogs (see figure 1-3 on page 31 and figure 1-4 on page 32 respectively). The button names (such as Job Control) are displayed when moving the mouse pointer over the buttons.

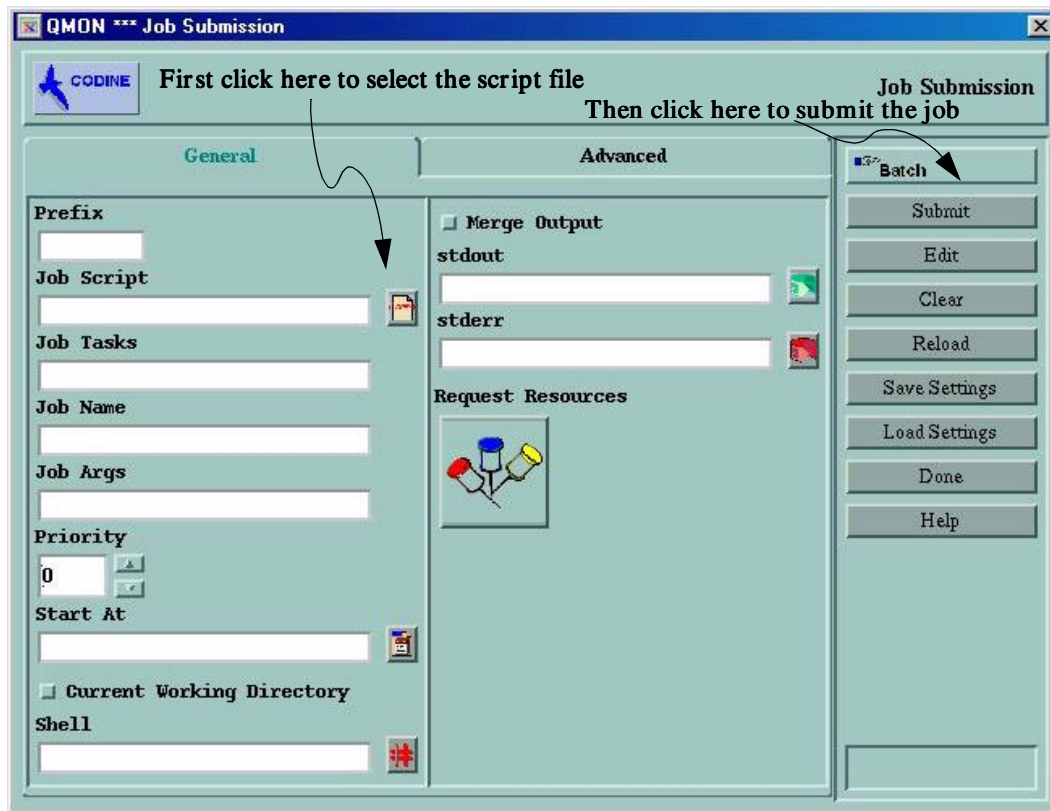


FIGURE 1-3 qmon Job Submission menu



FIGURE 1-4 qmon Job Control dialog

To submit a job from the Job Submission menu you first may want to select your job script file. Click on the Job Script file selection icon to open a file selection box and select your script file (e.g. the file *simple.sh* from the command line example). Then click on the Submit button at the bottom of the Job Submission menu. After a couple of seconds, you should be able to monitor your job in the Job Control panel. You will first see it under Pending Jobs and it will quickly move to Running Jobs once it gets started.

A Guide Through the Sun Grid Engine Manual Set

The *Sun Grid Engine Installation and Administration Guide*

The following central facilities and concepts of Sun Grid Engine installation and administration are presented in the *Sun Grid Engine Installation and Administration Guide*:

- Introduction

A short introduction of the Sun Grid Engine system and the Sun Grid Engine manual set is given.

- Installation

The installation procedure is described allowing for consideration of ample environmental conditions and site requirements. An overview on the directory structure generated and used by Sun Grid Engine is given.

- Architectural Dependencies

Differences for various operating system platforms are pointed out.

- Master and Shadow Master Configuration

The configuration of one or multiple hosts as Sun Grid Engine master hosts and failover servers for the master server is explained.

- Sun Grid Engine Daemons and Hosts

Properties and configuration of the various Sun Grid Engine host types are explained. It is also shown how Sun Grid Engine daemons can be shut down and restarted.

- Cluster Configuration

The cluster configuration contains cluster-wide and host-specific parameters defining filename paths used by Sun Grid Engine and specifying the general Sun Grid Engine behavior. The section explains these parameters and shows how to change them.

- **Configuring Queues**

Sun Grid Engine queues are the representations of the various job classes to be supported on a cluster. Thus, configuring queues is the projection of the intended utilization profile of the cluster onto Sun Grid Engine internal structures. The facilities to administer Sun Grid Engine queues are described in detail.

- **The Complexes Concept**

The *complexes concept* is central for the definition and handling of the attributes that jobs can request from the Sun Grid Engine system. Via the configuration of complexes user requestable resources are managed such as job limits, host load values, installed software, available software licenses and consumable resources like available memory. A detailed explanation of the concept and the handling of complexes is provided.

- **Queue Calendars**

Calendars allow to define availability and unavailability time periods for queues based on day-of-year, day-of-week and time-of-day. This chapter describes how to configure the queue availability policy for public holidays, weekends, office hours and the like.

- **Load Parameters**

Sun Grid Engine periodically retrieves a variety of system load and system information indices called load parameters from each execution host. These load parameters are used throughout the Sun Grid Engine system for load balancing and other load dependent scheduling policies. The section describes the standard load parameter set and explains the Sun Grid Engine interface to extend the standard set of load parameters by customized site specific indices.

- **Managing User Access**

Sun Grid Engine provides ample facilities to manage user permissions and access of users and user groups to the Sun Grid Engine system. This section describes the various categories of users in Sun Grid Engine, their configuration as well as the configuration and usage of access lists.

- **Scheduling**

The policies of a site with respect to resource utilization are mainly implemented by Sun Grid Engine's scheduling policies. The understanding of Sun Grid Engine's scheduling schemes and of the corresponding configuration facilities is vital to being able to implement policies. Therefore, this section contains a comprehensive description of the Sun Grid Engine scheduling and its configuration.

- **The Sun Grid Engine Path Aliasing Facility**

Sun Grid Engine provides a mechanism to hide file path inconsistencies which often occur in heterogeneous networked environments. Inconsistent file paths across hosts can be aliased to a single unique Sun Grid Engine internal name. The section explains when and how to use this facility.

- **Configuring Default Request**

The cluster administration can define default Sun Grid Engine job profiles for users to reduce the effort required for job submission and to avoid errors. The section explains how these so called default request can be used.

- **Setting Up a Sun Grid Engine User**

The required steps to set up Sun Grid Engine users are described. It is also explained how user access can be restricted.

- **Customizing Qmon**

The configuration capabilities of the OSF/Motif graphical user interface qmon are described.

- **Gathering Accounting and Utilization Statistics**

The accounting facilities of Sun Grid Engine is explained.

- **Checkpointing Support**

This section describes how Sun Grid Engine can utilize checkpointing environments and how Sun Grid Engine can be integrated with checkpointing facilities. It also gives an overview of the benefits, restrictions and prerequisites associated with checkpointing.

- **Support of Parallel Environments**

Sun Grid Engine provides a flexible and easy to use interface to arbitrary parallel environments including *shared memory*, PVM¹, MPI², etc. The configuration necessary to run parallel jobs on such environments is explained.

- **The Sun Grid Engine Queuing System Interface (QSI)**

Sun Grid Engine offers a general interface to other queuing systems. The steps for configuring the interface properly for exchanging jobs with an other queuing system are described.

- **Trouble Shooting**

Help is provided for the most common pitfalls and problems that may occur while installing, administering or running Sun Grid Engine.

The Sun Grid Engine User's Guide

- **Introduction**

A short introduction of the Sun Grid Engine system and the Sun Grid Engine manual set is given.

1.Parallel Virtual Machine, Oak Ridge National Laboratories

2.Message Passing Interface

- Sun Grid Engine User Types and Operations

The various user categories Sun Grid Engine supports and their permissions are explained.

- Navigating through the Sun Grid Engine System

Useful information is provided to the user for navigating through the Sun Grid Engine system and for retrieving required information.

- Submit Batch Jobs

Sun Grid Engine's job submission facilities are shown in many facets, among them: submission of batch, parallel and array jobs with `qmon` and from the command-line. The section explains in addition how a batch job should be constructed and how Sun Grid Engine interacts with the job script and schedules the job.

- Submit Interactive Jobs

Sun Grid Engine supports not only batch jobs but also interactive access to Sun Grid Engine resources. The differences between batch and interactive usage are explained.

- Transparent Remote Execution

Sun Grid Engine's means to transparently pass execution of specific tasks on to remote resources are described in this section. Sun Grid Engine provides its own versions of a remote shell command (`qrsh`), of a parallel make facility (`qmake`) and of an interactive command interpreter (`qtcsh`) for that purpose. This section is closely linked to the submission of interactive jobs.

- Checkpointing Jobs

Checkpointing Jobs are supported for fault tolerance and dynamic load balancing reasons. If they are aborted during execution because of system failure or because the job had to be removed from an overloaded system, the job can be migrated to another suitable host and can restart from the latest checkpoint. The preparations necessary to enforce checkpointing for a job are explained.

- Monitoring and Controlling Sun Grid Engine Jobs

Comprehensive facilities are available in Sun Grid Engine to monitor and control (e.g., cancel, suspend, resume) jobs and this section gives a detailed description.

- Job Dependencies

Sun Grid Engine jobs may depend on successful completion of other jobs running previously. The means how to set up job dependencies are described.

- Controlling Queues

The section shows how Sun Grid Engine queues can be monitored and controlled, i.e. suspended/resumed and disabled/enabled, via `qmon` and from the command-line.

- Customizing Qmon

The configuration capabilities of the OSF/Motif graphical user interface qmon are described.

The *Sun Grid Engine Reference Manual*

- User and Administrative Commands

All commands available to the user and administrator from the command line are described in detail including command line options, environmental conditions and the like. The command line invocation of graphical user's interfaces is also explained.

- Application Programmer's Interface

An interface and functionality description of the user callable Sun Grid Engine API routines is given.

- File Formats

The format of Sun Grid Engine administrative files is explained.

- Sun Grid Engine Daemons

All Sun Grid Engine daemons together with the feasible command line switches and environmental conditions are described for administrative purposes.

Glossary of Sun Grid Engine Terms

The glossary provides a short overview on frequently used terms in the context of Sun Grid Engine and resource management in general. Many of the terms have not been used so far, but will appear in other parts of the Sun Grid Engine documentation.

access list A list of users and UNIX groups who are permitted, or denied, access to a resource such as a queue or a certain host. Users and groups may belong to multiple access lists and the same access lists can be used in various contexts.

cell A separate Sun Grid Engine cluster with a separate configuration and master machine. Cells can be used to loosely couple separate administrative units.

| | |
|-----------------------------------|--|
| checkpointing | A procedure which saves the execution status of a job into a so called <i>checkpoint</i> thereby allowing for the job to be aborted and resumed later without loss of information and already completed work. The process is called <i>migration</i> , if the checkpoint is moved to another host before execution resumes. |
| checkpointing environment | A Sun Grid Engine configuration entity, which defines events, interfaces and actions being associated with a certain method of checkpointing. |
| cluster | A collection of machines, called hosts, on which Sun Grid Engine functions occur. |
| complex | A set of attributes that can be associated with a queue, a host, or the entire cluster. |
| group | A UNIX group. |
| hard resource requirements | The resources which must be allocated before a job may be started. Contrasts with <i>soft resource requirements</i> . |
| host | A machine on which Sun Grid Engine functions occur. |
| job | <p>A batch job is a UNIX shell script that can be executed without user intervention and does not require access to a terminal.</p> <p>An interactive job is a session started with the Sun Grid Engine commands <code>qsh</code> or <code>qlogin</code> that will open an <i>xterm</i> window for user interaction or provide the equivalent of a remote login session, respectively.</p> |
| job array | A job consisting of a range of independent identical tasks. Each task is very similar to a separate job. Job array tasks only differ by a unique task identifier (an integer number). |
| job class | A set of jobs that are equivalent in some sense and treated similarly. In Sun Grid Engine a job class is defined by the identical requirements of the corresponding jobs and the characteristics of the queues being suitable for those queues. |
| manager | A user who can manipulate all aspects of Sun Grid Engine. The superusers of the master host and of any other machine being declared as an administrative host have manager privileges. Manager privileges can be assigned to non-root user accounts as well. |
| migration | The process of moving a checkpoint from one host to another before execution of the job resumes. |

| | |
|-----------------------------------|--|
| operator | Users who can perform the same commands as managers except that they cannot change the configuration but rather are supposed to maintain operation. |
| owner | Users who may suspend/unsuspend and disable/enable the queues they own. Typically users are owners of the queues that reside on their workstations. |
| parallel environment | A Sun Grid Engine configuration entity, which defines the necessary interfaces for Sun Grid Engine to correctly handle parallel jobs. |
| parallel job | A job which consists of more than one closely correlated task. Tasks may be distributed across multiple hosts. Parallel jobs usually use communication tools such as shared memory or message passing (MPI, PVM) to synchronize and correlate tasks. |
| policy | A set of rules and configurations which the Sun Grid Engine administrator can use define the behavior of Sun Grid Engine. Policies will be implemented automatically by Sun Grid Engine. |
| priority | The relative level of importance of a Sun Grid Engine job compared to others. |
| queue | A container for a certain class and number of jobs being allowed to execute on a Sun Grid Engine execution host concurrently. |
| resource | A computational device consumed or occupied by running jobs. Typical examples are memory, CPU, I/O bandwidth, file space, software licenses, etc. |
| soft resource requirements | Resources which a job needs but which do not have to be allocated before a job may be started. Allocated to a job on an as available basis. Contrast with <i>hard resource requirements</i> . |
| suspension | The process of holding a running job but keeping it on the execution machine (in contrast to checkpointing, where the job is aborted). A suspended job still consumes some resources, such as swap memory or file space. |
| user | May submit jobs to and execute jobs with Sun Grid Engine if s/he has a valid login on at least one submit host and an execution host. |
| userset | Either an access list (see above) or a department (see above). |

Installation and Administration Guide

Introduction

Sun Grid Engine (Computing in Distributed Networked Environments) is a *load management* tool for heterogeneous, distributed computing environments. Sun Grid Engine provides an effective method for distributing the batch workload among multiple computational servers. In doing so, it increases the productivity of all of the machines and simultaneously increases the number of jobs that can be completed in a given time period. Also, by increasing the productivity of the workstations, the need for outside computational resources is reduced.

Please refer to the *Sun Grid Engine Quick Start Guide* for an overview on the Sun Grid Engine system, its features and components. The *Sun Grid Engine Quick Start Guide* also contains a quick installation procedure for a small sample Sun Grid Engine configuration and a glossary of terms commonly used in the Sun Grid Engine manual set.

For detailed information on the end-user related aspects of Sun Grid Engine, the reader is pointed to the *Sun Grid Engine User's Guide*. In addition, the *Sun Grid Engine Reference Manual* provides reference manual pages for all Sun Grid Engine commands, components and file formats.

The remainder of the *Sun Grid Engine Installation and Administration Guide* will focus on detailed installation instructions and a comprehensive description of Sun Grid Engine's administrative tasks and toolsets.

Installation

Overview

Installation consists of:

- Planning the Sun Grid Engine configuration and environment.
- Reading the Sun Grid Engine distribution files from an external medium onto a workstation.
- Running an installation script on the master host and every execution host in the Sun Grid Engine system.
- Registering information about administrative and submit hosts.
- Verifying the installation.

Installation should be done by someone familiar with UNIX. It is done in three phases:

Phase 1 - Planning

1. **Decide whether your Sun Grid Engine environment will be a single cluster or a collection of sub-clusters called *cells*.**
2. **Select the machines that will be Sun Grid Engine hosts. Determine what kind(s) of host(s) each machine will be — master host, shadow master host, administration host, submit host and/or execution host.**
3. **Make sure that all Sun Grid Engine users have the same user ids on all submit and execution hosts.**
4. **Decide what the Sun Grid Engine directory organization will be (for example, a complete tree on each workstation, directories cross mounted, a partial directory tree on some workstations) and where each Sun Grid Engine root directory will be located.**
5. **Decide on the site's queue structure.**
6. **Decide whether network services will be defined as an NIS (Network Information Services) file or local to each workstation in `/etc/services`.**
7. **Complete the installation worksheet (refer to table 2-1 on page 49). You will use this information in subsequent installation steps.**

Phase 2 - Install the Software

1. Create the installation directory and load the distribution files into it.
2. Install the master host.
3. Install all execution hosts.
4. Register all administrative hosts.
5. Register all submit hosts.

Phase 3 - Verify the Installation

1. Check that the daemons are running on the master host.
2. Check that the daemons are running on all execution hosts.
3. Check that Sun Grid Engine executes simple commands.
4. Submit test jobs.

Planning

Prerequisite Tasks

The Sun Grid Engine installation procedure creates a default configuration for the system it is executed on. It inquires the operating system type hosting the installation and makes meaningful settings based on this information.

The following sections contain the information to install a production Sun Grid Engine system:

The Installation Directory <codine_root>

Please prepare a directory to read in the contents of the Sun Grid Engine distribution media. This directory will be called the Sun Grid Engine *root directory* and later-on, while the Sun Grid Engine system is in operation, will be used to store the current cluster configuration and all further data that needs to be spooled to disk.

By default, <codine_root> is located in /usr/CODINE. If this is changed, Sun Grid Engine administrators or users must set the environment variable CODINE_ROOT to the new location before running commands. Use a path name that is a correct

reference on all hosts. For example, if the file system is mounted using automounter, set CODINE_ROOT to /usr/CODINE, not /tmp_mnt/usr/CODINE. Throughout this document we will use *<codine_root>* when referencing the installation directory.

<codine_root> is the top level of the Sun Grid Engine directory tree. Each Sun Grid Engine component in a cell (see section “Cells” on page 48) needs read access to *<codine_root>/<cell>/common* on start up. The *root* user on the master and shadow master hosts also need write access to this directory.

For ease of installation and administration it is recommended that this directory is readable on all hosts you intend to execute the Sun Grid Engine installation procedure on. You may, for example, select a directory available via a network file system (like NFS). If you choose to select filesystems local to the hosts you will have to copy the installation directory to each host before you start the installation procedure for the particular machine.

Spool Directories Under the Root Directory

On the Sun Grid Engine master host spool directories are maintained under *<codine_root>/<cell>/spool/qmaster* and *<codine_root>/<cell>/spool/schedd*.

On each execution host a spool directory called *<codine_root>/<cell>/spool/<exec_host>* is maintained. This directory does not need to be exported to other machines.

Directory Organization

Decide what the Sun Grid Engine directory organization will be (for example, a complete tree on each workstation, directories cross mounted, a partial directory tree on some workstations) and where each Sun Grid Engine root directory, *<codine_root>*, will be located.

Note – Since a change of the installation directory and/or the spool directories basically requires a new installation of the system (although all important information from the previous installation can be preserved), the user should carefully select a suitable installation directory upfront.

By default, the Sun Grid Engine installation procedure will install the Sun Grid Engine system, manuals, spool areas and the configuration files in a directory hierarchy (figure 2-1 on page 45) under the installation directory. If you accept this default behavior, you should install/select a directory which allows the access permissions described in section “File Access Permissions”.

You can move the spool areas to other locations after the primary installation (see section “Cluster Configuration” on page 70 for the required parameter configuration changes). You are also free to move the binaries, libraries and the manuals (subdirectories `bin`, `lib`, `man` and `doc`) to arbitrary locations (just make sure that the search and manual path variables are set properly).

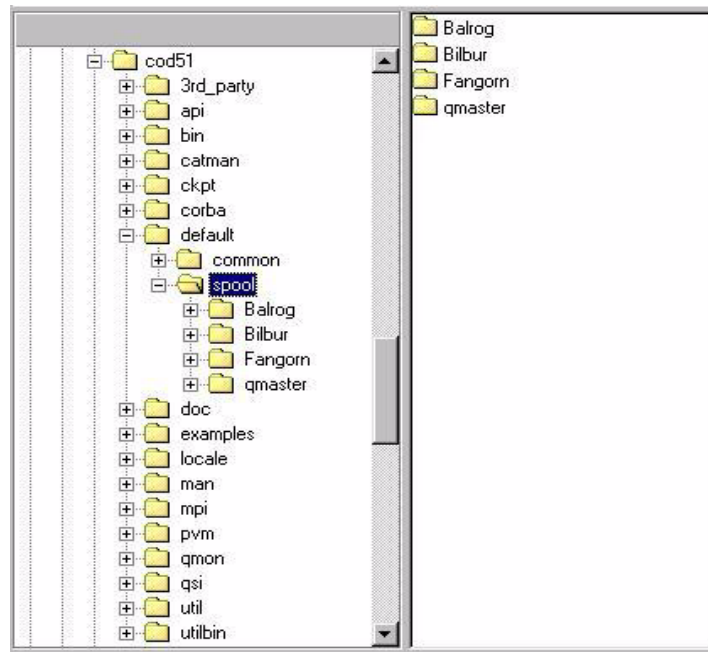


FIGURE 2-1 Sample directory hierarchy

Disk Space Requirements

The Sun Grid Engine directory tree has certain fixed disk space requirements. They are:

- 40 MB for the installation kit (including documentation) without any binaries.
- Between 10 and 15 MB for each set of binaries, except for the architecture Cray, where the binaries consume approximately 35 MB.

In addition, we recommend providing the following disk space for Sun Grid Engine log files

- 30-200 MB for the master host spool directories depending on the size of the cluster
- 10-20 MB for each execution host.

Note – The spool directories of the master host and the execution hosts are configurable and do not have to reside under `<codine_root>` (where they are located by default). Changing the location of the spool directories should be done after the primary installation (see section “Cluster Configuration” on page 70).

Installation Accounts

You have the possibility to install Sun Grid Engine either under the *root* account or under an unprivileged (e.g. your own) account. The consequence of installing under an unprivileged account is that this installation will only allow for that particular user to run Sun Grid Engine jobs. Access will be denied to all other accounts. Installing under the *root* account resolves this restriction, however *root* permission is required for the complete installation procedure.

File Access Permissions

If you install as *root*, you may have a problem to configure root read/write access for all hosts on a shared file system and thus you may have problems to put `<codine_root>` onto a network wide file system. You can force Sun Grid Engine to run the entire file handling of all Sun Grid Engine components through a non-root administrative user account (called *codine*, for example). Thus you only need read/write access to the shared root file system for this particular user. The Sun Grid Engine installation procedure will ask whether you want file handling under an administrative user account. Only if you answer with **Yes** and provide a valid *user id*, file handling will be performed via this user id. Otherwise, the user id under which you run the installation procedure will be used.

You have to make sure in all cases that the account used for file handling has read/write access on all hosts to the Sun Grid Engine root directory. Also, the installation procedure assumes that the host from which you will read in the Sun Grid Engine distribution media can access this directory.

Network Services

Determine whether your site’s network services are defined as a NIS (Network Information Services) file or local to each workstation in `/etc/services`. If your site uses NIS, find out the NIS server host so that you can add entries to the services NIS map.

Master Host

This is the host from which Sun Grid Engine is controlled. It runs the master daemon, `cod_qmaster`. The master host is central to Sun Grid Engine's operation, so it should:

- be a stable platform,
- not be excessively busy with other processing,
- have at least 20 Mbytes of unused main memory to run the Sun Grid Engine daemons,
- (optional) have the Sun Grid Engine directory, `<codine_root>`, local to it to cut down on network traffic.

Shadow Master Hosts

These hosts back up `cod_qmaster`'s functionality in case the master host or the master daemon fails. To be a shadow master host, a machine must:

- run `cod_shadowd`.
- share `cod_qmaster`'s status, job, and queue configuration information that is logged to disk. In particular, the shadow master hosts need read/write root access to the `cod_qmaster`'s spool directory and to the `<codine_root>/<cell>/common` directory.
- the `<codine_root>/<cell>/common/shadow_masters` file contains a line defining the host as a shadow master host.

The shadow master host facility is activated for a host as soon as these conditions are met. So you do not need to restart Sun Grid Engine daemons to make a host into a shadow host

Execution Hosts

These hosts run the jobs that are submitted to Sun Grid Engine. You will run an installation script on each execution host.

Administrative Hosts

Sun Grid Engine operators and managers perform administrative tasks such as reconfiguring queues or adding Sun Grid Engine users from these hosts. The master host installation script automatically makes the master host an administrative host.

Submit Hosts

Sun Grid Engine jobs may be submitted and controlled from submit hosts. The master host installation script automatically makes the master host a submit host.

Cells

You may set up Sun Grid Engine as a single cluster or a collection of loosely coupled clusters called *cells*. The `COD_CELL` environment variable indicates the cluster being referenced. When Sun Grid Engine is installed as a single cluster, `COD_CELL` is not set and the value `default` is assumed for the cell value.

User Ids

In order for Sun Grid Engine to verify that users submitting jobs have permission to submit them and to use the execution hosts they need, users ids must be identical on the submit and execution hosts involved. This requirement may necessitate changing user ids on some machines.

Note – The user ids on the master host are not relevant for permission checking and do not have to match or even do not have to exist.

Queues

Plan the queue structure that meets your site's needs. This means determining what queues should be placed on which execution hosts, whether you need queues for sequential, interactive, parallel and other job types, how many job slots are needed in each queue, and other queue configuration decisions.

It is also possible for the Sun Grid Engine administrator to let the installation procedure create a default queue structure, which is suitable for getting acquainted with the system and as starting point for later tuning.

Note – Despite the directory Sun Grid Engine is installed to, all settings created by the Sun Grid Engine installation procedure can be changed during operation of the system on the fly.

In case you are already familiar with Sun Grid Engine or you previously have decided on the queue structure you want to impose on your cluster, you should not allow the installation procedure to install a default queue structure for you. But

instead, you should prepare a document specifying that queue structure and you should proceed to section "Configuring Queues" on page 75 directly after completing the installation process.

Installation Plan

Please write down your installation plan in a table similar to the one included below before you begin with the installation.

| Parameter | Value |
|----------------------|-------|
| codine_root | |
| master host | |
| shadow master hosts | |
| execution hosts | |
| administrative hosts | |
| submit hosts | |

TABLE 2-1 Template form to be filled in prior to Installation

You should now ensure that the file system(s) and directories that will contain the Sun Grid Engine distribution and the spool and configuration files are set up properly. Please set the access permissions as defined above.

Reading the Distribution Media

Sun Grid Engine is distributed either on CD-ROM or as archive file through Internet download. Please ask your system administrator or refer to your local system documentation for how to access CD-ROMs. The CD-ROM distribution contains a

file with a *tape archive* (tar format) and several README files for direct access. The Web distribution is also provided in tar file format eventually compressed with compress (extension .z) or with gzip (extension .gz). Please uncompress the file (use `uncompress` or `gunzip`) before proceeding with the next step.

Provide access to the distribution media and login to a system preferably with direct connection to a file server. Create the installation directory as described in section “The Installation Directory <codine_root>” to read in the Sun Grid Engine installation kit. Make sure that the access permissions for the installation directory are set properly.

Now, execute the following procedure from the command prompt:

```
% cd install_dir
% tar -xvpf distribution_source
```

where *install_dir* is the pathname of the installation directory and *distribution_source* is either the name of the uncompressed tape archive file on hard disk or CD-ROM. This will read in the Sun Grid Engine installation kit.

Installing the Master Host

Login to the master host as *root*. If the directory where the installation kit resides is visible from the master host, `cd` to the installation directory. If the directory is not visible and cannot be made visible, create a local installation directory on the master host and copy the installation kit to the local installation directory via the network (e.g. by using `ftp` or `rcp`). Afterwards `cd` to the local installation directory. Now execute the following instruction:

```
% ./inst_codine -m
```

This will initiate the master installation procedure. You will be asked several questions and may be forced to execute some administrative actions. The questions and the action items should be self-explanatory.

Note – It is recommended to have a second terminal session active to execute administrative tasks.

The master installation procedure creates the appropriate directory hierarchy required by `cod_qmaster` and `cod_schedd`. The procedure starts up the Sun Grid Engine components `cod_commd`, `cod_qmaster` and `cod_schedd` on the master host. The master host is also registered as host with administrative and submit permission.

If you feel that something went wrong you can abort and repeat the installation procedure at any time.

Installing Execution Hosts

In order to start the execution host installation, login as *root* to the execution host. As for the master installation either copy the installation kit to a local installation directory or use a network installation directory. `cd` to the installation directory and execute:

```
% ./inst_codine -x
```

This will initiate the execution host installation procedure. The behavior and handling of the execution host installation procedure is very similar to the one for the master host. Please follow the same directions as given in section "Installing the Master Host" on page 50.

Note – You may use the master host also for execution of jobs. You just need to carry out the execution host installation for the master machine.

Note – If you use a very slow machine as master host and/or if your cluster is considerably large, it is recommended to use the master machine for the master task only.

The execution host installation procedure creates the appropriate directory hierarchy required by `cod_execd`. The procedure starts up the Sun Grid Engine components `cod_commd` and `cod_execd` on the execution host.

Installing Administration and Submit Hosts

The master host is implicitly allowed to execute administrative tasks and to submit, monitor and delete jobs. It does not require any kind of additional installation as administration or submit host. As opposed to this, *pure* administration and submit hosts simply require registration with the commands:

```
% qconf -ah admin_host_name[,...]  
% qconf -as submit_host_name[,...]
```

The commands need to be executed from an administrative host (e.g. the master host) and by an administrative account (e.g. the super user account).

Please refer to section "Sun Grid Engine Daemons and Hosts" on page 56 for more details and other means to configure the different host types.

Verifying the Installation

First make sure that the Sun Grid Engine daemons are running. In order to look for the `cod_qmaster`, `cod_schedd` and `cod_commd` daemons on the master machine, login to the master host and execute the UNIX command `ps -ax` if the master host runs a BSD based UNIX or `ps -ef` if the master host's UNIX is SYSV based. Parse through the output of `ps` and look for the string `cod_qmaster`. If you do not find lines (in the BSD case) looking for example like:

```
14673 p1 S <    2:12 /usr/CODINE/bin/sun4/cod_commd  
14676 p1 S <    4:47 /usr/CODINE/bin/sun4/cod_qmaster  
14678 p1 S <    9:22 /usr/CODINE/bin/sun4/cod_schedd
```

or (in the SYSV case) like:

```
root 439 1 0 Jun 22 ? 3:37 /usr/CODINE/bin/sgi/cod_commd  
root 442 1 0 Jun 22 ? 3:37 /usr/CODINE/bin/sgi/cod_qmaster  
root 446 1 0 Jun 22 ? 3:37 /usr/CODINE/bin/sgi/cod_schedd
```

one or multiple of the Sun Grid Engine daemons required on the master host are not running on this machine (you can look into the file `<codine_root>/<cell>/common/act_qmaster_name` whether you really are on the master host). You can try to restart the daemons by hand. Section "Sun Grid Engine Daemons and Hosts" on page 56 describes how to proceed.

In order to look for the daemons required on the execution machines, login to the execution hosts the Sun Grid Engine execution host installation procedure was run on. Again execute `ps` and look for the string `cod_execd` in the output. If you do not find lines like (in the BSD case):

```
14685 p1 S < 1:13 /usr/CODINE/bin/sun4/cod_commd
14688 p1 S < 4:27 /usr/CODINE/bin/sun4/cod_execd
```

or (in the SYSV case) like:

```
root 169 1 0 Jun 22 ? 2:04 /usr/CODINE/bin/sgi/cod_commd
root 171 1 0 Jun 22 ? 7:11 /usr/CODINE/bin/sgi/cod_execd
```

one or multiple daemons required on the execution host are not running. Again section "Sun Grid Engine Daemons and Hosts" on page 56 describes how to restart the daemons by hand.

If both the necessary daemons run on the master and execution hosts the Sun Grid Engine system should be operational. You can check if Sun Grid Engine accepts commands by simply typing:

```
% qconf -sconf
```

from the command line when logged into either the master host or another administrative host (do not forget to include the path where you installed the Sun Grid Engine binaries into your standard search path). This `qconf` command displays the current global cluster configuration (see section "Cluster Configuration" on page 70).

If this command fails, most probably either your `CODINE_ROOT` environment variable is set inappropriately or `qconf` fails to contact the `cod_commd` associated with `cod_qmaster`. In this case, you should check whether the script files `<codine_root>/<cell>/common/settings.csh` or `<codine_root>/<cell>/common/settings.sh` set the environment variable `COMMD_PORT`. If so, please make sure that the environment variable `COMMD_PORT` is set to that particular value before you try the above command again. If the `COMMD_PORT` variable is not used in the settings files, the services database (e.g. `/etc/services` or the NIS services map) on the machine you executed the command must provide a `cod_commd` entry. If this is not the case, please add such an entry to the machine's services database and give it the same value as is configured on the Sun Grid Engine master host. Then retry the `qconf` command.

Before you start submitting batch scripts to the Sun Grid Engine system, please check if your sites standard and your personal shell resource files (`.cshrc`, `.profile` or `.kshrc`) contain inconvenient commands like `setty` (batch jobs do not

have a terminal connection by default and, therefore, calls to `stty` will result in an error). An easy way to do this is to login to the master host and to execute the command:

```
% rsh an_exec_host date
```

an_exec_host means one of the already installed execution hosts you are going to use (you should check on all execution hosts if your login and/or home directories differ from host to host). The `rsh` command should give you an output very similar to the `date` command executed locally on the master host. If there are any additional lines containing error messages, the reasons for the errors must be removed prior to be able to run a batch job successfully.

For all command interpreters you can check on an actual terminal connection before you execute a command like `tty`. The following is a Bourne-/Korn-Shell example how to do this:

```
tty -s
if [ $? = 0 ]; then
    stty erase ^H
fi
```

The C-Shell syntax is very similar:

```
tty -s
if ( $status = 0 ) then
    stty erase ^H
endif
```

Note – The leading `tty -s` is an exception as it causes no problems with batch execution.

Now you are ready to submit batch jobs. First you should try to submit one of the example scripts contained in the directory `<codine_root>/examples/jobs`. To submit them, just use the command:

```
% qsub script_path
```

and use the Sun Grid Engine `qstat` command to monitor the job's behavior (please refer to the *Sun Grid Engine User's Guide* for more information about submitting and monitoring batch jobs). As soon as the job has finished execution please check your home directory for the redirected stdout/stderr files `<script_name>.e<job_id>` and `<script_name>.<job_id>` with `<job_id>` being a consecutive unique integer number assigned to each job.

In case of problems, please see section "Trouble Shooting" on page 158.

Architectural Dependencies

Any difference in functionality depending on the operating system architecture Sun Grid Engine runs on is documented in files starting with the string `arc_depend_` in the directory `<codine_root>/doc`. The remainder of the file name indicates the operating system architectures to which the comments in the files apply.

Master and Shadow Master Configuration

The shadow master hostname file `<codine_root>/<cell>/common/shadow_masters` contains the name of the primary master host (the machine the Sun Grid Engine master daemon `cod_qmaster` is initially running on) and the so called shadow master hosts. The format of the master hostname file is as follows:

- The first line of the file defines the primary master host.
- The following lines specify the shadow master hosts, one per line.

The order of appearance of the (shadow) master hosts is significant. If the primary master host (the first line in the file) fails to proceed, the shadow master defined in the second line will take over. If this one fails also, the one defined in the third line is on duty and so forth.

In order to prepare a host as Sun Grid Engine shadow master the following requirements must be met:

- A shadow master host needs to run `cod_shadowd`.
- The shadow master hosts need to share `cod_qmaster`'s status information, job and queue configuration logged to disk. In particular the (shadow) master hosts need read/write root access to the master's spool directory and to the directory `<codine_root>/<cell>/common`.

- The shadow master hostname file has to contain a line defining the host as shadow master host.

As soon as these requirement are met, the shadow master host facility is activated for this host. No restart of Sun Grid Engine daemons is necessary to activate the feature.

The automatic failover start of a `cod_qmaster` on a shadow master host will take some time (in the order of one minute). Meanwhile you will get a corresponding error message whenever a Sun Grid Engine command is executed.

Note – The file `<codine_root>/<cell>/common/act_qmaster` contains the name of the host actually running the `cod_qmaster` daemon.

In order to be able to start a shadow `cod_qmaster` Sun Grid Engine must be sure that either the *old* `cod_qmaster` has terminated or that it will terminate without performing actions interfering with the just started shadow `cod_qmaster`. Under very rare circumstances this is impossible. In these cases a corresponding error message will be logged to the messages logfile of the `cod_shadowds` on the shadow master hosts (see section “Trouble Shooting” on page 158) and any attempts to open a `tcp` connection to a `cod_qmaster` daemon will permanently fail. If this occurs, Please make sure, that no master daemon is running and restart `cod_qmaster` manually on any of the shadow master machines (see section “Killing and Restarting Daemons” on page 69).

Sun Grid Engine Daemons and Hosts

Classification

Sun Grid Engine hosts are classified into four groups, depending on which Sun Grid Engine daemons are running on the system and how the hosts are registered at `cod_qmaster`:

1. Master host:

The master host is central for the overall cluster activity. It runs the master daemon `cod_qmaster`. `cod_qmaster` controls all Sun Grid Engine components such as queues and jobs and maintains tables about the status of the components, about user access permissions and the like. Section “Installation” on page 42 describes how to initially set up the master host and section “Master and Shadow Master Configuration” on page 55 shows how dynamic master host changes can

be configured. The master host usually runs the Sun Grid Engine scheduler `cod_schedd`. The master host requires no further configuration other than performed by the installation procedure.

2. Execution hosts:

Execution hosts are nodes having permission to execute Sun Grid Engine jobs. Therefore, they are hosting Sun Grid Engine queues and run the Sun Grid Engine execution daemon `cod_execd`. An execution host is initially set up by the execution host installation procedure as described in section "Installing Execution Hosts" on page 51).

3. Administration hosts:

Permission can be given to other hosts than the master host to carry out any kind of administrative activity in Sun Grid Engine. Administrative hosts are set up with the command `qconf -ah hostname` (see the `qconf` manual page for details).

4. Submit hosts:

Submit hosts allow for submitting and controlling batch jobs only. In particular a user being logged into a submit host can submit jobs via `qsub`, can control the job status via `qstat` or run Sun Grid Engine's OSF/1 Motif graphical user's interface `qmon`. Submit hosts are set up with the command `qconf -as hostname` (see the `qconf` manual page for details)

Note – A host may belong to more than one of the above described classes.

Note – The master host is an administrative and submit host by default.

Configuring Hosts

Sun Grid Engine maintains object lists for all types of hosts except for the master host. In the case of the administrative and submit hosts these lists simply provide the information whether or not a host has administrative or submit permission. In the case of the execution host object, further parameters, such as the load information as reported by the `cod_execd` running on the host is stored there as well as load parameter scaling factors to be provided by the Sun Grid Engine administrator.

The following sections explain how to configure the different host objects with the help of the Sun Grid Engine OSF/Motif graphical user's interface `qmon` and from the command-line.

The GUI administration is provided by a set of host configuration dialogues which are invoked by pushing the `Host Config` icon button in the `qmon` main menu. The available dialogues are the administration host configuration (see figure 2-2), the submit host configuration (see figure 2-3) and the execution host configuration (see figure 2-4). The dialogues can be switched by using the selection list button at the top of the screen.

The `qconf` command provides the command-line interface for the host object management.

Administrative Hosts

The *Administration Host Configuration* dialogue is opened upon selecting `Administration Host` in the tab widget on the top of the screen. The administration host configuration dialogue is opened by default when the `Host Config` button is pressed for the first time.

With this dialogue hosts can be declared from which administrative Sun Grid Engine commands are allowed. The selection list in the center of the screen displays the hosts already declared to provide administrative permission. An existing host can be deleted from this list by clicking on its name with the left mouse button and by pushing the `Delete` button at the bottom of the dialogue. A new host can be added by entering its name to the `Hostname` input window and pressing the `Add` button afterwards.

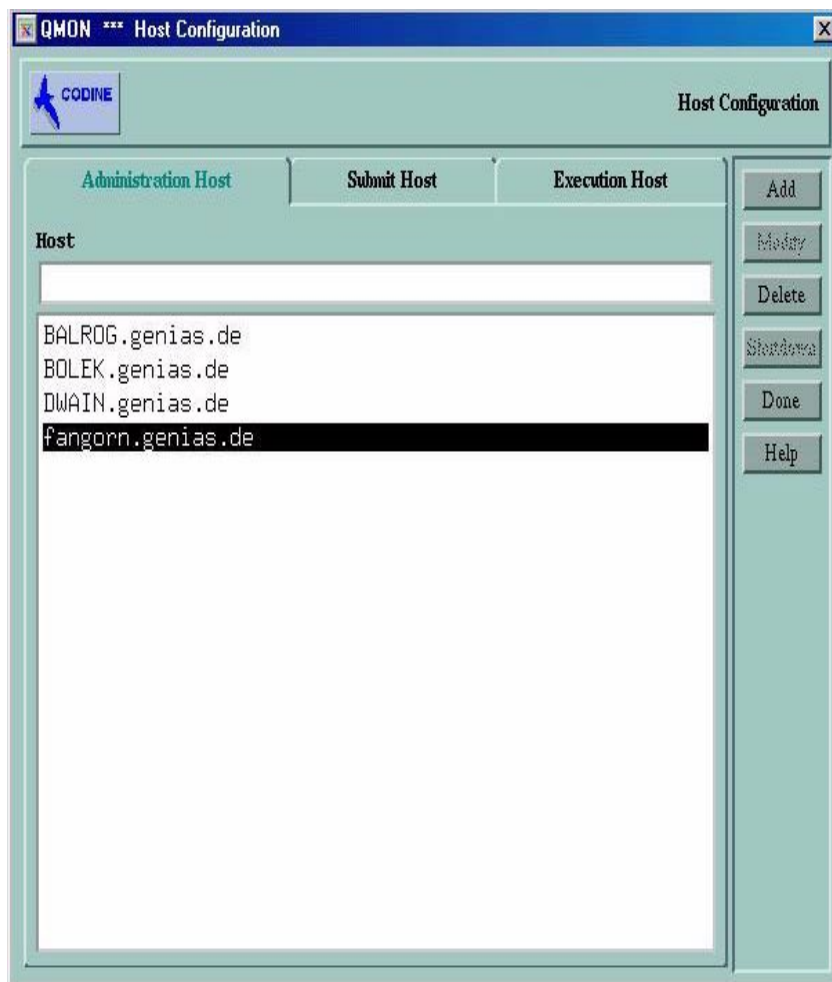


FIGURE 2-2 Administration Host Configuration

The command-line interface for maintaining the list of administration hosts is provided by the following options to the `qconf` command:

`qconf -ah hostname`

add administrative host. Adds the specified host to the list of administrative hosts.

`qconf -dh hostname`

delete administrative host. Deletes the specified host from the list of administrative hosts.

```
qconf -sh
```

show administrative hosts. Displays a list of all currently configured administrative hosts.

Submit Hosts

The *Submit Host Configuration* dialogue is opened upon selecting `Submit Host` in the tab widget on the top of the screen. Hosts can be declared from which jobs can be submitted, monitored and controlled. No administrative Sun Grid Engine commands are allowed from these hosts unless they are declared to be administrative hosts also (see “Administrative Hosts” on page 58). The selection list in the center of the screen displays the hosts already declared to provide submit permission. An existing host can be deleted from this list by clicking on its name with the left mouse button and by pushing the `Delete` button at the bottom of the dialogue. A new host can be added by entering its name to the `Hostname` input window and pressing the `Add` button afterwards.

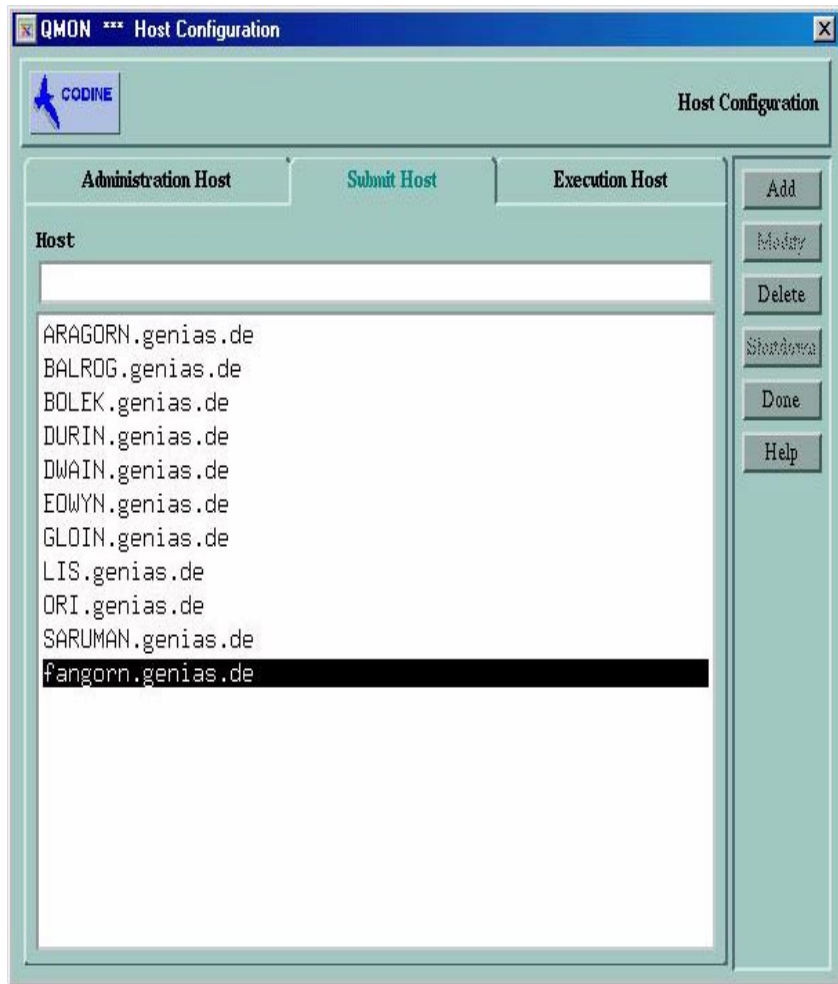


FIGURE 2-3 Submit Host Configuration

The command-line interface for maintaining the list of submit hosts is provided by the following options to the `qconf` command:

`qconf -as hostname`

add submit host. Adds the specified host to the list of submit hosts.

`qconf -ds hostname`

delete submit host. Deletes the specified host from the list of submit hosts.

```
qconf -ss
```

show submit hosts. Displays a list of the names of all hosts currently configured to provide submit permission.

Execution Hosts

The *Execution Host Configuration* dialogue is opened upon selecting *Execution Host* in the tab widget on the top of the screen. Sun Grid Engine execution hosts can be configured from this dialogue. No administrative or submit commands are automatically allowed from these hosts unless they are declared to be administrative or submit hosts also (see “Administrative Hosts” on page 58 and “Submit Hosts” on page 60).

The *Hosts* selection list displays the execution hosts already defined. The currently configured load scaling factors, the access permissions and the resource availability for consumable and fixed complex attributes associated with the host are displayed in the *Load Scaling*, the *Access Attributes* and the *Consumable/Fixed Attributes* display windows for the selected execution host. Please refer to section “The Complexes Concept”, section “User Access Permissions” and section “Load Parameters”, for details on complex attributes, user access permissions and load parameters.

An existing host can be deleted from the list of execution hosts by clicking on its name with the left mouse button and by pushing the *Delete* button at the button column on the right side of the dialogue. The execution daemon *cod_execd* on an execution host can be shut down by pushing the *Shutdown* button for any selected host. A new host can be added or modified pushing the *Add* or *Modify* button in the button column. This will open the dialogue displayed in figure 2-5 on page 65 and described below.

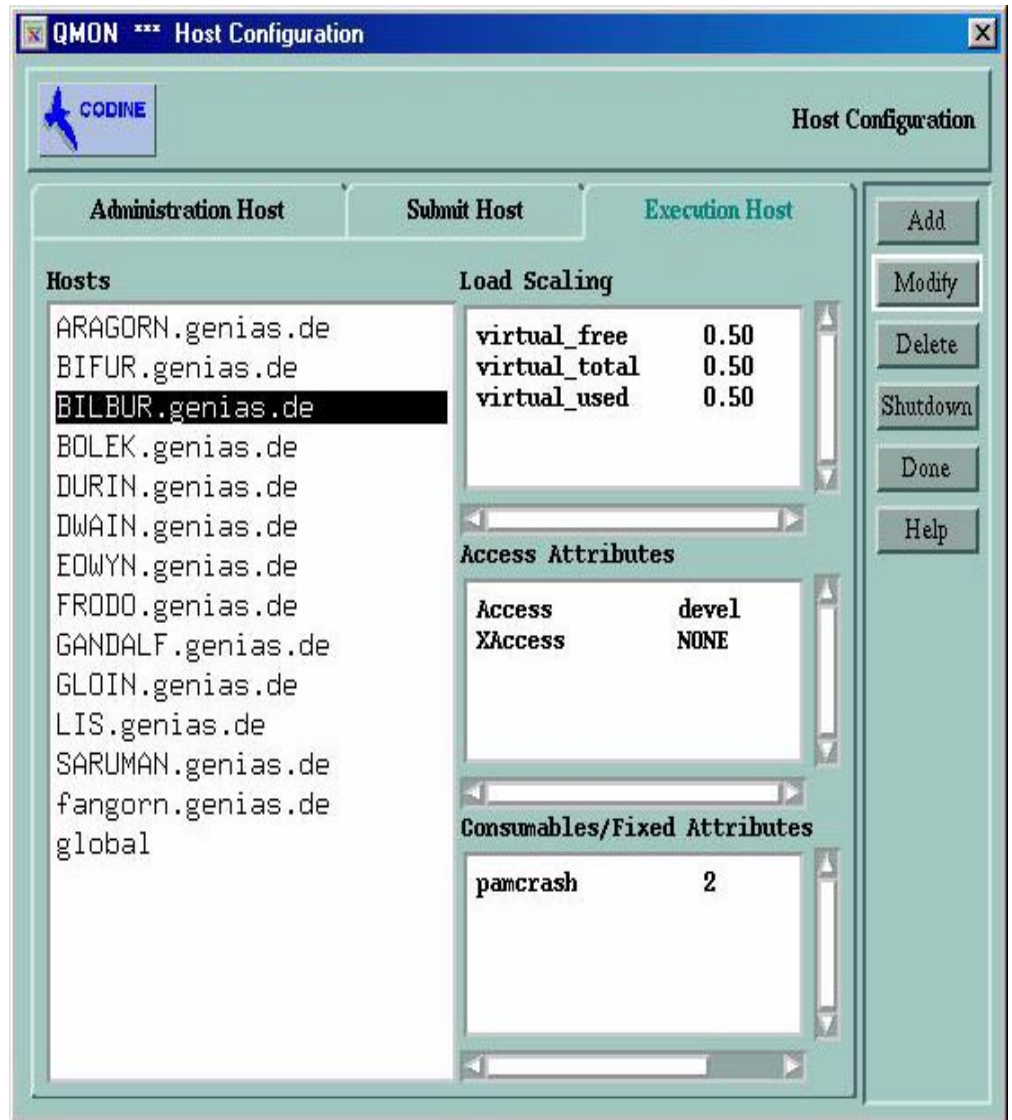


FIGURE 2-4 Execution Host Configuration

The dialogue to add a new execution host or modify the configuration of an existing one allows for modification of all attributes associated with the host. The name of the execution host is displayed or can be added in the Host input window. Scaling factors can be defined if Scaling is selected in the tab widget of the dialogue (see figure 2-5 on page 65).

All available load parameters are displayed in the Load column of the Load Scaling table and the corresponding definition of the scaling can be found in the Scale Factor column. The Scale Factor column can be edited. Valid scaling factors are positive floating point numbers in fixed point or scientific notation.

If Consumables/Fixed Attributes is selected in the tab widget, the complex attributes associated with the host can be defined (see figure 2-7 on page 66). The complexes (see section “The Complexes Concept”) associated with the host are the *global* and the *host complex* or the *administrator defined complexes* attached to the host via the Complex Selection area on the left bottom of the dialogue. Available administrator defined complexes are displayed on the left and they can be attached or detached via the red arrows. The Complex Config icon button opens the top level complex configuration dialogue in case you need further information on the current complex configuration or if you want to modify it.

The Consumable/Fixed Attributes table in the right bottom area of the dialogue enlists all complex attributes for which a value currently is defined. The list can be enhanced by clicking to the Name or Value button at the top. This will open a selection list with all attributes attached to the host (i.e. the union of all attributes configured in the global, the host and the administrator defined complexes attached to this host as described above). The attribute selection dialogue is shown in figure 2-7 on page 66. Selecting one of the attributes and confirming the selection with the Ok button will add the attribute to the Name column of the Consumable/Fixed Attributes table and will put the pointer to the corresponding Value field. Modifying an existing value can be achieved by double-clicking with the left mouse button on the Value field. Deleting an attribute is performed by first selecting the corresponding table line with the left mouse button. The selected list entry can be deleted either by typing CTRL-D or by clicking the right mouse button to open a deletion box and confirming the deletion.

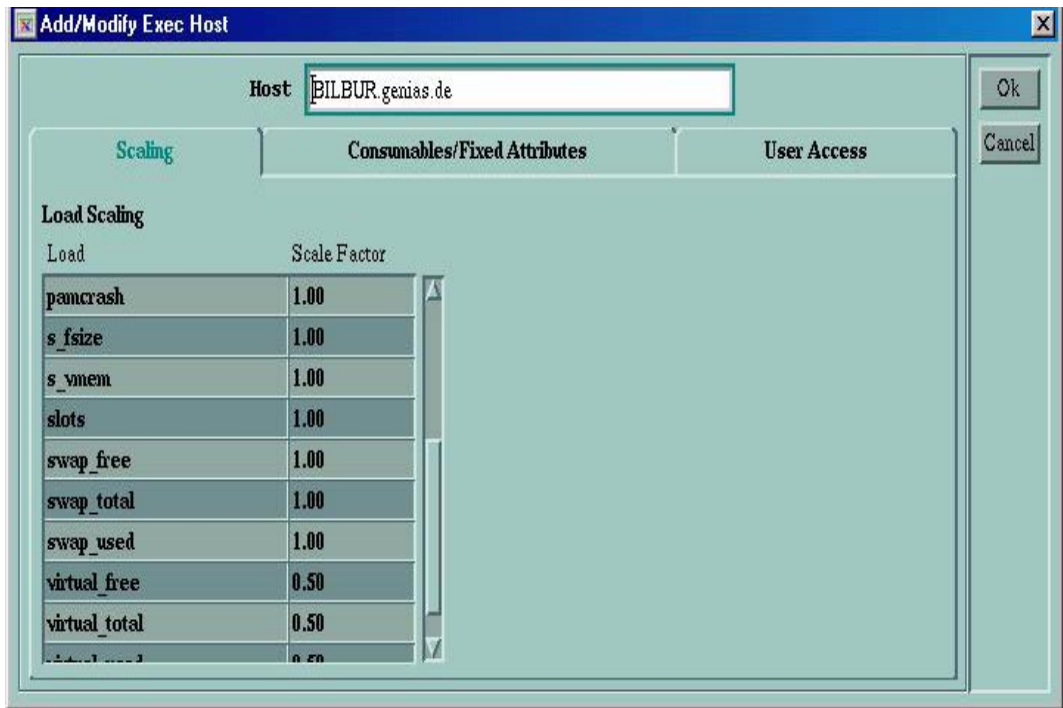


FIGURE 2-5 Modify Load Scaling

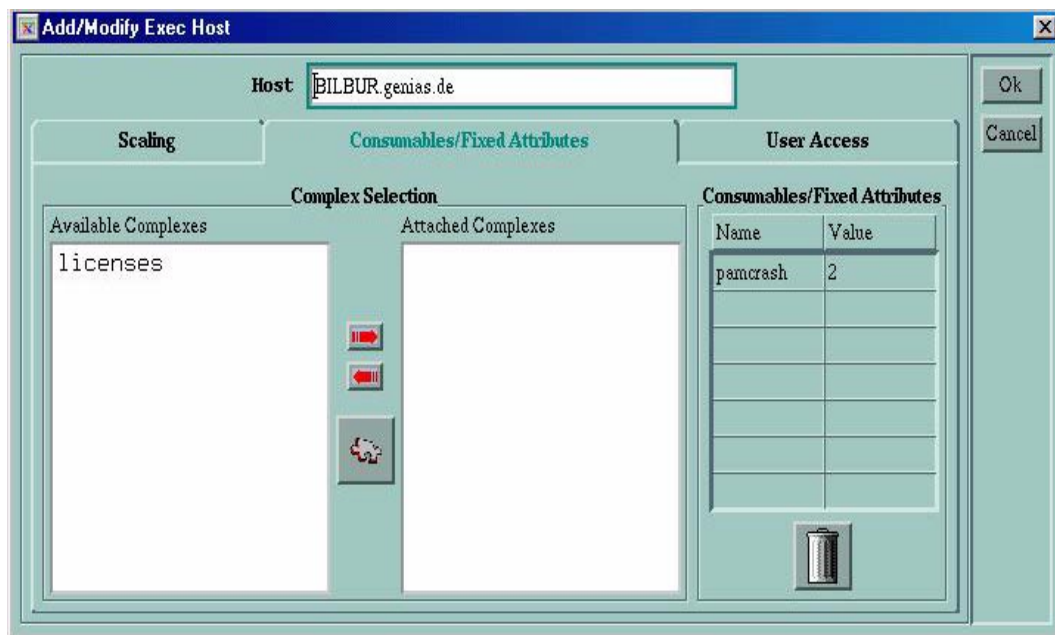


FIGURE 2-6 Modify Consumable/Fixed Attributes



FIGURE 2-7 Available complex attributes

If User Access is selected in the tab widget (figure 2-8 on page 67), the access permissions to the execution host can be defined based on previously configured user access lists (section “Configure User Access Lists with qmon”).

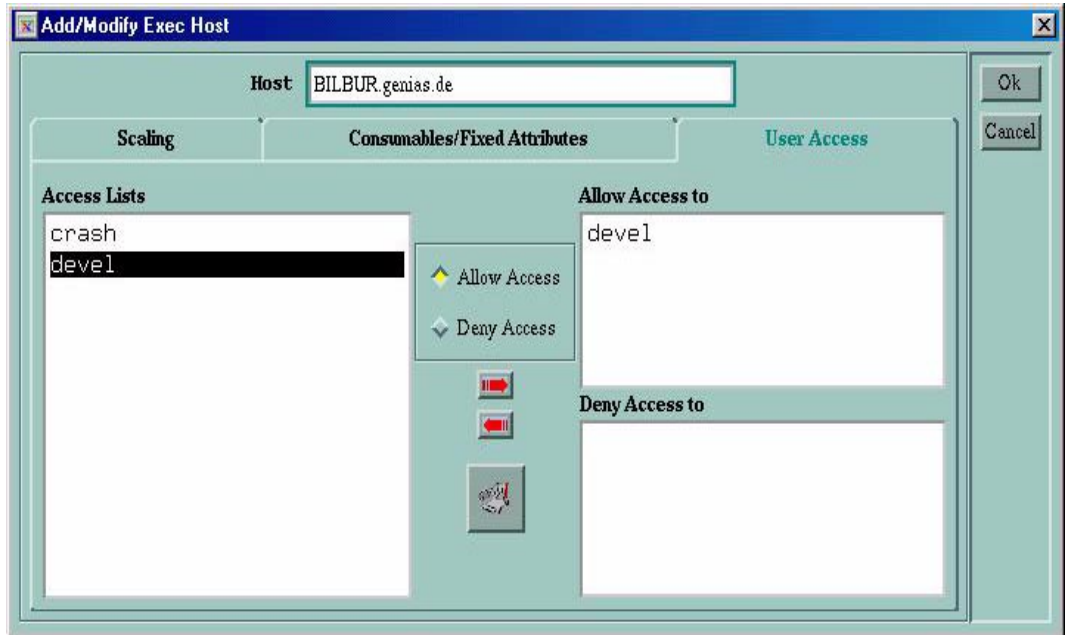


FIGURE 2-8 Modify User Access

The command-line interface for maintaining the list of execution hosts is provided by the following options to the `qconf` command:

```
qconf -ae [exec_host_template]
```

add execution host. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with an execution host configuration template. If the optional parameter `exec_host_template` (the name of an already configured execution host) is present the configuration of this execution host is used as template. The execution host is configured by changing the template and saving to disk. See the `host_conf` manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.

```
qconf -de hostname
```

delete execution host. Deletes the specified host from the list of execution hosts. All entries in the execution host configuration are lost.

`qconf -me hostname`

modify execution host. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with the configuration of the specified execution host as template. The execution host configuration is modified by changing the template and saving to disk. See the `host_conf` manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.

`qconf -Me filename`

modify execution host. Uses the content of *filename* as execution host configuration template. The configuration in the specified file must refer to an existing execution host. The configuration of this execution host is replaced by the file content. This `qconf` option is useful for off-line execution host configuration changes, e.g. in cron jobs, as it requires no manual interaction.

`qconf -se hostname`

show execution host. Show the configuration of the specified execution host as defined in `host_conf`.

`qconf -sel`

show execution host list. Display a list of host names which are configured to be execution hosts.

Monitoring Execution Hosts with qghost

The `qghost` command provides a convenient way to retrieve a quick overview on the execution host status. Various options are provided to customize the information retrieved and the output format displayed.

In its standard form:

`% qghost`

an output similar to the following will be printed:

TABLE 2-2 Sample qghost Output

| HOSTNAME | ARCH | NPROC | LOAD | MEMTOT | MEMUSE | SWAPTO | SWAPUS |
|-------------------|----------|-------|------|--------|--------|--------|--------|
| global | - | - | - | - | - | - | - |
| BALROG.genias.de | solaris6 | 2 | 0.38 | 1.0G | 994.0M | 900.0M | 891.0M |
| BILBUR.genias.de | solaris | 1 | 0.18 | 96.0M | 70.0M | 164.0M | 9.0M |
| DWAIN.genias.de | irix6 | 1 | 1.13 | 149.0M | 55.8M | 40.0M | 0.0 |
| GLOIN.genias.de | osf4 | 2 | 0.05 | 768.0M | 701.0M | 1.9G | 13.5M |
| SPEEDY.genias.de | alinux | 1 | 0.08 | 248.8M | 60.6M | 125.7M | 232.0K |
| SARUMAN.genias.de | solaris | 1 | 0.11 | 96.0M | 77.0M | 192.0M | 9.0M |
| FANGORN.genias.de | linux | 1 | 2.01 | 124.8M | 49.9M | 127.7M | 4.3M |

Please refer to the qghost manual page in the *Sun Grid Engine Reference Manual* for a description of the output format and for further options.

Killing and Restarting Daemons

In order to immediately halt the Sun Grid Engine system on your cluster you can use the commands:

```
% qconf -kej
% qconf -ks
% qconf -km
```

The first command will kill all currently active jobs and bring down all Sun Grid Engine execution daemons.

Note – If replacing that command by `qconf -ke`, the Sun Grid Engine execution daemons are aborted, but the active jobs are not cancelled. Jobs which finish while no `cod_execd` is running on that system are not reported to `cod_qmaster` until `cod_execd` is restarted again. The job reports are not lost, however.

The second command will shutdown the Sun Grid Engine scheduler `cod_schedd`. The third command finally will force the `cod_qmaster` process to terminate. You will need Sun Grid Engine manager or operator privileges for these operations (see section "Managing User Access" on page 117).

If you have running jobs and you want to wait with the shutdown procedure of Sun Grid Engine until the currently active jobs are finished you can use the command below for each queue before executing the `qconf` sequence described above.

```
% qmod -d queue_name
```

The `qmod` disable command prevents new jobs from being scheduled to the disabled queues. You should then wait with the killing of the daemons until no jobs run in the queues any longer.

To restart daemons on a particular machine you will have to login to that machine as *root* and execute the procedure:

```
% <codine_root>/<cell>/common/codine5
```

This script will look for the daemons normally running on this host and subsequently start the corresponding ones.

Cluster Configuration

The Basic Cluster Configuration

The basic Sun Grid Engine cluster configuration is a set of information configured to reflect site dependencies like valid paths for programs such as `mail` or `xterm` and to influence the Sun Grid Engine behavior. There is a global configuration, which is provided by for the Sun Grid Engine master host as well as every host in the Sun Grid Engine pool. In addition, the Sun Grid Engine system may be configured to use a configuration local to every host to override particular entries in the global configuration.

The `sge_conf` manual page in the *Sun Grid Engine Reference Manual* contains a detailed description of the configuration entries. The Sun Grid Engine cluster administrator should adapt the global and local configurations to the site's needs directly after the installation and keep it up to date afterwards.

Displaying the Basic Cluster Configurations

The Sun Grid Engine command to display the current configuration is the `show` configuration option of the `qconf` program. The following are a few examples (see the *Sun Grid Engine Reference Manual* for a detailed description):

```
% qconf -sconf
% qconf -sconf global
% qconf -sconf <host>
```

The first two commands are equivalent and will display the global configuration. The third command will display the host's local configuration.

Modifying the Basic Cluster Configurations

The Sun Grid Engine command to change the cluster configurations may be used by Sun Grid Engine administrators only. Examples for such commands are:

```
% qconf -mconf global
% qconf -mconf <host>
```

The first command example will modify the global configuration while the second example operates on the local configuration of the specified execution or master host.

Displaying the Cluster Configuration with qmon

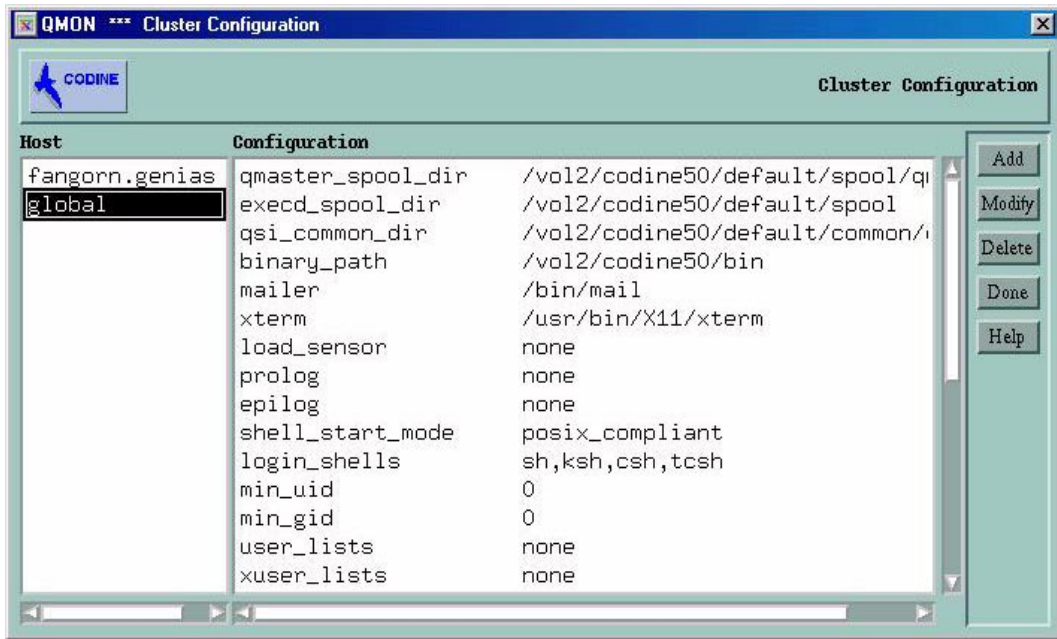


FIGURE 2-9 Cluster Configuration dialogue

The Cluster Configuration dialogue as displayed in figure 2-9 on page 72 is opened by clicking with the left mouse button on the Cluster Configuration icon button in the qmon main menu. By selecting a particular configuration for a host in the Host selection list on the left side of the screen, the dialogue can be used to display the current configuration for that host or to delete the selected configuration when pressing the Delete button. Selecting the special name `global` in the host selection list displays the global configuration.

The configurations are displayed in the format which is described in the `sgc_conf` manual page. Use the button `Modify` to modify the selected global or host local configuration. Use the `Add` button to add a new configuration for a specific host.

Modifying global and Host Configurations with qmon

The image shows a screenshot of the 'Cluster Settings' dialog box, specifically the 'General Settings' tab. The dialog is titled 'Cluster Settings' and has a subtitle 'Configuration for Host'. A text field at the top shows 'global'. The 'General Settings' tab is active, and the 'Advanced Settings' tab is also visible. The 'General Settings' section contains several fields for configuration parameters. The 'Master Spool Dir' is set to '/vol2/codine50/default/spool/qmaster'. The 'Execd Spool Dir' is set to '/vol2/codine50/default/spool'. The 'QSI Common Dir' is set to '/vol2/codine50/default/common/qsi'. The 'Binary Path' is set to '/vol2/codine50/bin'. The 'Mailer' is set to '/bin/mail'. The 'Xterm' is set to '/usr/bin/X11/xterm'. The 'Load Sensor' is set to 'none'. The 'Admin User' is set to 'none'. The 'Admin Mail' is set to 'none'. The 'Prolog' is set to 'none'. The 'Epilog' is set to 'none'. The 'Login Shells' are set to 'sh,ksh,csh,tcsh'. The 'Shell Start Mode' is set to 'posix_compliant'. The 'Log Level' is set to 'log_info'. The 'Advanced Settings' section contains fields for 'Min UID', 'Min GID', and 'Finished Jobs'. 'Min UID' is set to 0, 'Min GID' is set to 0, and 'Finished Jobs' is set to 50. There are also fields for 'Loadreport Time' (00:01:00) and 'Max Unheard' (00:02:30). The 'Stat Log Time' is set to 12:00:00. There are also sections for 'User Lists' and 'Xuser Lists'.

| Parameter | Value |
|------------------|--------------------------------------|
| Master Spool Dir | /vol2/codine50/default/spool/qmaster |
| Execd Spool Dir | /vol2/codine50/default/spool |
| QSI Common Dir | /vol2/codine50/default/common/qsi |
| Binary Path | /vol2/codine50/bin |
| Mailer | /bin/mail |
| Xterm | /usr/bin/X11/xterm |
| Load Sensor | none |
| Admin User | none |
| Admin Mail | none |
| Prolog | none |
| Epilog | none |
| Login Shells | sh,ksh,csh,tcsh |
| Shell Start Mode | posix_compliant |
| Log Level | log_info |
| Min UID | 0 |
| Min GID | 0 |
| Finished Jobs | 50 |
| Loadreport Time | 00:01:00 |
| Max Unheard | 00:02:30 |
| Stat Log Time | 12:00:00 |

FIGURE 2-10 Cluster Settings dialogue General Settings

The Cluster Settings dialogue is opened upon clicking to the Modify or Add button in the Cluster Configuration dialogue described in section "Displaying the Cluster Configuration with qmon" on page 72. It provides the means for changing all parameters of a global or host local configuration. All entry fields are only accessible if the global configuration is changed, i.e. if the the selected host was global and if Modify was pressed. If a regular host is modified, its actual configuration is reflected in the dialog and only those parameters can be modified which are feasible for host local changes. If a new host local configuration is added, the dialogue entries will be empty fields.

The Advanced Settings tab (figure 2-11 on page 74) shows a corresponding behavior depending on whether a global, host local or new configuration is changed. It provides access to more rarely used cluster configuration parameters.

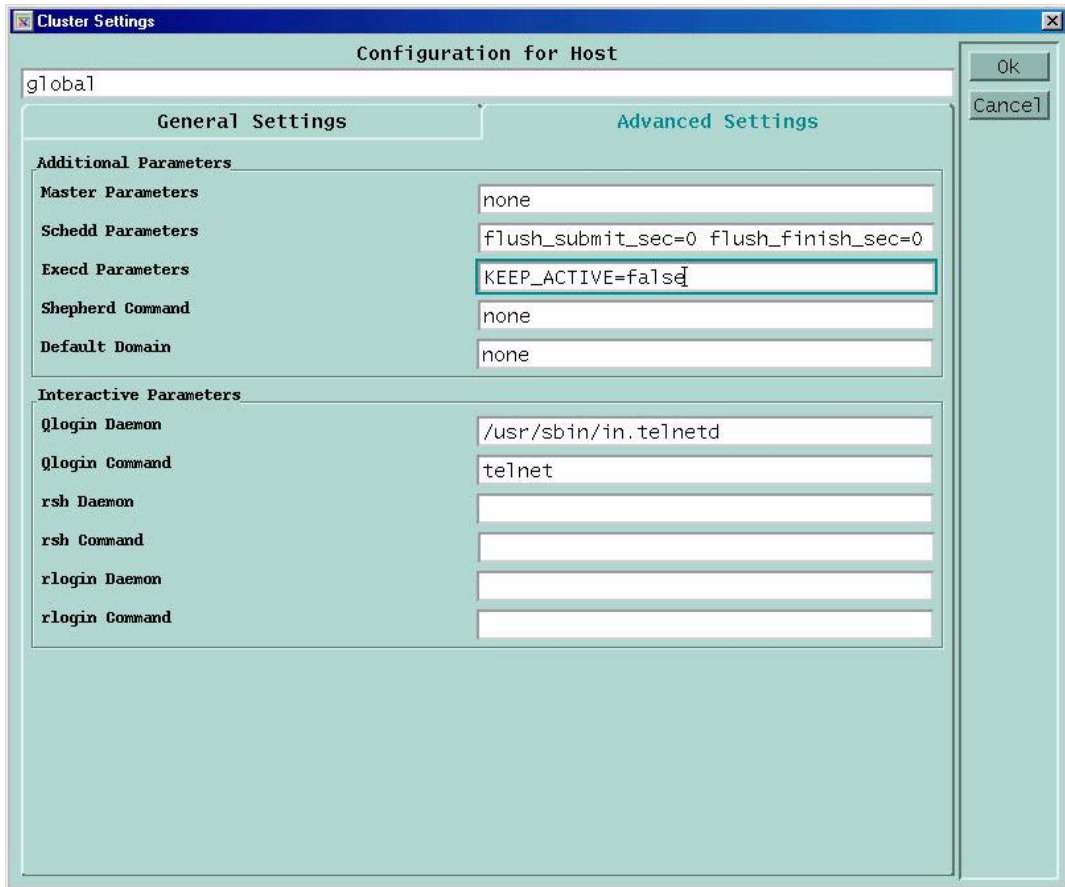


FIGURE 2-11 Cluster Settings dialogue Advanced Settings

After finishing the modifications, the Ok button on the right upper corner will register the modified configuration. Pressing Cancel discards any changes. The dialogue is closed in both cases.

Please refer to the `sge_conf` manual page for a complete description of all cluster configuration parameters.

Configuring Queues

Sun Grid Engine queues are containers for different categories of jobs and provide the corresponding resources for concurrent execution of multiple jobs belonging to the same category. Jobs will not wait in Sun Grid Engine queues but start running immediately as soon as they are dispatched. The Sun Grid Engine scheduler's job pending list is the only waiting area for Sun Grid Engine jobs.

Configuring Sun Grid Engine queues will register the queue attributes with `cod_qmaster`. As soon as they are configured, they are instantly visibly to the whole cluster and to all Sun Grid Engine users on all hosts belonging to the Sun Grid Engine pool.

Configuring Queues with `qmon`

The Queue Configuration dialogue is opened upon pushing the Add or Modify button in the Queue Control dialogue. The Queue Control dialogue and its facilities to monitor and manipulate the queue status are described in section "Controlling Queues with `qmon`" on page 232 of the *Sun Grid Engine User's Guide*.

If the Queue Configuration dialogue is opened for the first time it shows the general parameters form (see "Configuring General Parameters" on page 76).

The queue to be affected by the desired operation is displayed or defined in the Queue and Hostname windows in the upper screen region. If a queue is to be modified an existing queue has to be selected in the Queue Control dialogue before the Queue Configuration dialogue is opened. A queue name and a host on which the queue resides must be defined if a new queue is going to be added.

To increase the ease of use of the Queue Configuration dialogue, three buttons are available directly below the Hostname window: The Clone button, which allows for the import of all parameters of an existing queue via a queue selection list, the Reset button, which loads the configuration of the template queue and the Refresh button, which loads the configuration of other objects which were modified while the Queue Configuration dialogue was open (see section "Queue Configuration "User Complexes"" on page 83 and "Queue Configuration "User Access" parameters" on page 85 for further details concerning the Refresh button).

The Ok button on the right upper corner of the Queue Configuration dialogue registers the changes with `cod_qmaster`, while the Cancel button below discards any changes. Both buttons close the dialogue.

Nine parameter sets are available to define a queue: *General* (see "Configuring General Parameters" on page 76), *Execution Method* (see section "Configuring Execution Method Parameters" on page 77), *Checkpointing* (see "Configuring Execution Method Parameters" on page 77), *Load/Suspend Thresholds* (see "Configuring Load and Suspend Thresholds" on page 79), *Limits* (see "Configuring Limits" on page 81), *Complexes* (see "Configuring User Complexes" on page 82), *Subordinates* ("Configuring Subordinate Queues" on page 84), *User Access* (see "Configuring User Access" on page 85) and *Owners* (see "Configuring Owners" on page 86). The desired parameter set can be selected via the queue parameter tab widget.

Configuring General Parameters

If the General parameter set is selected, the parameter set definition region looks as displayed in figure 2-12 below. The fields offered allow for setting the following parameters:

- Sequence number of the queue.
- Processors - a specifier for the processor set to be used by the jobs running in that queue. For some operating system architectures this can be a range (s.th. like 1-4,8,10) or just an integer identifier of the processor set. See the `arc_depend_*.asc` files in the `doc` directory of your Sun Grid Engine distribution for more information.
- Temporary directory path.
- Default command interpreter (Shell) to be used to execute the job scripts.
- A calendar attached to the queue defining *on-duty* and *off-duty* times for the queue.
- The time waited between delivery of SIGUSR1/SIGUSR2 notification signals and suspend/kill signals (Notify).
- The nice value with which to start the jobs in this queue (0 means use system default).
- The number of jobs to be allowed to execute concurrently in the queue (job slots).
- The type of the queue and of the jobs being allowed to execute in this queue. Multiple selections are feasible.
- The Shell Start Mode, i.e. the mode in which to start the job script.
- The Initial State in which a newly added queue comes up or in which the queue is restored if the `cod_execd` running on the queue host gets restarted.
- The queue's default rerun policy to be enforced on jobs which have been aborted e.g. due to system crashes. The user may overwrite this policy by the `qsub -r` option or the job submission dialogue (see figure 3-8 on page 181 of the *Sun Grid Engine User's Guide*).

Please refer to the `queue_conf` manual page for detailed information on these parameters.

Queue Configuration: Modify

Queue: gloin.q

Hostname: GLOIN.genias.de

Clone Reset Refresh

Complexes Subordinates User Access Owners

General Configuration Execution Method Checkpointing Load/Suspend Thresholds Limits

Sequence Nr: 0

Processors: UNDEFINED

tmp Directory: /tmp

Shell: /bin/csh

Calendar: weekend-night

Notify Time: 00:00:60

Job's Nice: 0

Slots: 4

Shell Start Mode: posix_compliant Initial State: default

Rerun Jobs: ☒

Type

- ☒ Batch
- ☒ Interactive
- ☒ Checkpointing
- ☐ Parallel
- ☐ Transfer

FIGURE 2-12 Queue Configuration “General” parameters

Configuring Execution Method Parameters

If the Execution Method parameter set is selected, the parameter set definition region looks as displayed in figure 2-13 below. The fields offered allow for setting the following parameters:

- A queue specific prologue and epilogue script executed with the same environment as the job before the job script is started and after it is finished respectively.
- A start/suspend/resume/terminate method overwriting Sun Grid Engine’s default methods for these applying these actions to jobs.

Please refer to the `queue_conf` manual page for detailed information on these parameters.

FIGURE 2-13 Queue Configuration “Execution Method” parameters

Configuring Checkpointing Parameters

If the Checkpointing parameter set is selected, the parameter set definition region looks as displayed in figure 2-14 below. The fields offered allow for setting the following parameters:

- The load thresholds initiating a job migration. A threshold value can be supplied for any load parameter. The currently configured thresholds are displayed in the Migration Load Thresholds box. An existing threshold can be selected and changed by double-clicking with the left mouse button to the corresponding Value field. To add new thresholds click to the Name or Value button at the top. This will open a selection list with all attributes attached to the queue (see “The Complexes Concept” on page 88 for details). The attribute selection dialogue is shown in figure 2-7 on page 66. Selecting one of the attributes and confirming the selection with the Ok button will add the attribute to the Name column of the Migration Load Thresholds table and will put the pointer to the corresponding Value field. A selected list entry can be deleted either by typing CTRL-D or by clicking the right mouse button to open a deletion box and confirming the deletion.

- The time waited between a checkpoint signal and a kill signal (MaxMigrTime). If set, enforces a checkpoint being generated at the time of migration.
- The allowed time for a checkpointing job to be spent outside checkpointing applications (MaxNoMigr).
- The periodical checkpoint interval (MinCpuTime).

Please refer to the `queue_conf` manual page for detailed information on these parameters.

Queue Configuration: Modify

Queue:

Hostname:

Complexes | **Subordinates** | **User Access** | **Owners**

General Configuration | **Execution Method** | **Checkpointing** | **Load/Suspend Thresholds** | **Limits**

Migration Load Thresholds:

| Load | Value |
|-------------|-------|
| np_load_avg | 5.00 |
| | |
| | |
| | |
| | |
| | |
| | |

MaxMigrTime:

MaxNoMigr:

MinCpuTime:

FIGURE 2-14 Queue Configuration “Checkpointing” parameters

Configuring Load and Suspend Thresholds

If the Load/Suspend Thresholds parameter set is selected, the parameter set definition region looks as displayed in figure 2-15 below. The fields offered allow for setting the following parameters:

- The Load Thresholds and the Suspend Thresholds tables, which define overload thresholds for load parameters and consumable complex attributes (see “The Complexes Concept” on page 88). Overload in the case of load thresholds results in preventing the queue from receiving further jobs by Sun Grid Engine. Exceeding one or more suspend thresholds causes suspension of jobs in the queue

to reduce the load. The currently configured thresholds are displayed in the tables. An existing threshold can be selected and changed by double-clicking with the left mouse button to the corresponding Value field. To add new thresholds click to the Name or Value button at the top. This will open a selection list with all valid attributes attached to the queue. The attribute selection dialogue is shown in figure 2-7 on page 66. Selecting one of the attributes and confirming the selection with the Ok button will add the attribute to the Name column of the corresponding threshold table and will put the pointer to its Value field. A selected list entry can be deleted either by typing CTRL-D or by clicking the right mouse button to open a deletion box and confirming the deletion.

- The number of jobs which are suspended per time interval to reduce the load on the system which hosts the configured queue.
- The time interval between suspension of further jobs in case suspend thresholds are still exceeded.

Please refer to the `queue_conf` manual page for detailed information on these parameters.

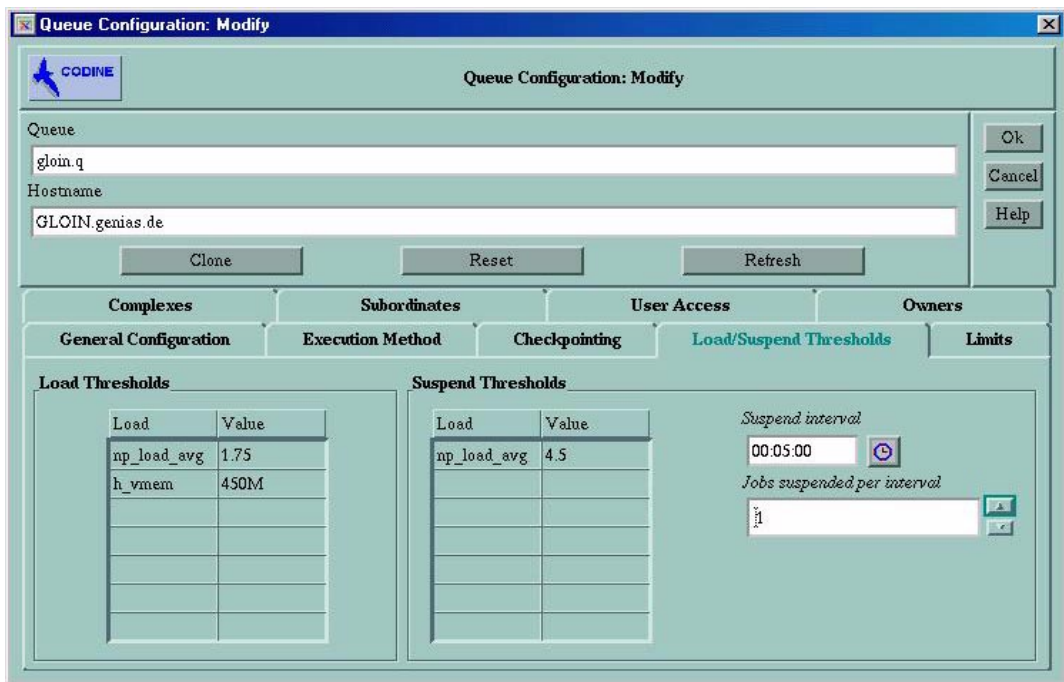


FIGURE 2-15 Queue Configuration “Load Thresholds”

Configuring Limits

If the **Limits** parameter set is selected, the parameter set definition region looks as displayed in figure 2-16 below. The fields offered allow for setting the following parameters:

- The *hard* and *soft* limits which are to be imposed on the jobs running in the queue.

To change a value of a limit double-click to the **Value** field of the limit entry. Double clicking to a **Value** field twice opens convenient input dialogues for either **Memory** or **Time** limit values (see figure 2-17 and figure 2-18).

Please refer to the `queue_conf` and `setrlimit` manual page for detailed information on the individual limit parameters and their interpretation for different operating system architectures.

Queue Configuration: Modify

Queue: gloin.q

Hostname: GLOIN.genias.de

Clone Reset Refresh

Ok Cancel Help

Complexes Subordinates User Access Owners

General Configuration Execution Method Checkpointing Load/Suspend Thresholds Limits

Double click on values to change limits

| Hard Limit | Value |
|--------------------------|----------|
| Wallclock Time (sec) | 24:00:00 |
| CPU Time (sec) | 12:00:00 |
| File Size (Byte) | INFINITY |
| Data Size (Byte) | INFINITY |
| Stack Size (Byte) | INFINITY |
| Corefile Size (Byte) | INFINITY |
| Resident Set Size (Byte) | 500M |

| Soft Limit | Value |
|--------------------------|----------|
| Wallclock Time (sec) | INFINITY |
| CPU Time (sec) | 08:00:00 |
| File Size (Byte) | INFINITY |
| Data Size (Byte) | INFINITY |
| Stack Size (Byte) | INFINITY |
| Corefile Size (Byte) | INFINITY |
| Resident Set Size (Byte) | INFINITY |

FIGURE 2-16 Queue Configuration “Limits”

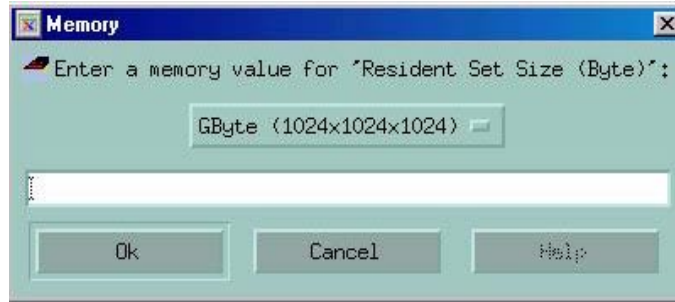


FIGURE 2-17 Memory input dialogue

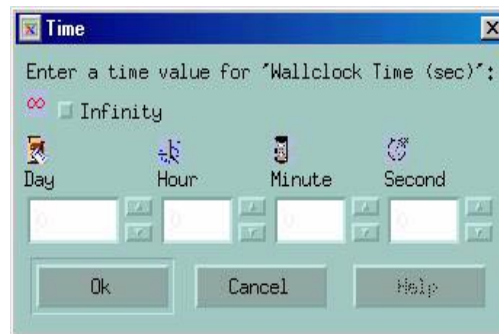


FIGURE 2-18 Time input dialogue

Configuring User Complexes

If the User Complexes parameter set is selected, the parameter set definition region looks as displayed in figure 2-19 below. The fields offered allow for setting the following parameters:

- The set of user defined complexes (see “User Defined Complexes” on page 93) being attached to the queue. The red arrows in the center of the Complex Selection box allow to attach and detach a user defined complex from/to the queue.
- A value definition for selected attributes from the set of complexes parameters available for this queue. The available complex parameters are assembled per default from the global complex, the host complex and from the attached user defined complexes. Attributes are either consumable or fixed parameters. The definition of a queue value defines a capacity managed by the queue in the case of a consumable attribute or simply a fixed, queue specific value in the case of fixed attributes (see section “The Complexes Concept” on page 88 for further details). The attributes, for which values are explicitly defined are displayed in the Consumable/Fixed Attributes table. An existing attribute can be selected and changed by double-clicking with the left mouse button to the

corresponding Value field. To add new attribute definitions click to the Name or Value button at the top. This will open a selection list with all valid attributes attached to the queue. The attribute selection dialogue is shown in figure 2-7 on page 66. Selecting one of the attributes and confirming the selection with the Ok button will add the attribute to the Name column of the attribute table and will put the pointer to its Value field. A selected list entry can be deleted either by typing CTRL-D or by clicking the right mouse button to open a deletion box and confirming the deletion.

Please refer to the queue_conf manual page for detailed information on these parameters.

The Complex Configuration dialogue (see “Complex Configuration dialogue “licenses”” on page 94 for example) is opened upon clicking on the Complex Config icon button. The current complexes configuration can be checked or modified before user defined complexes are attached or detached to a queue.

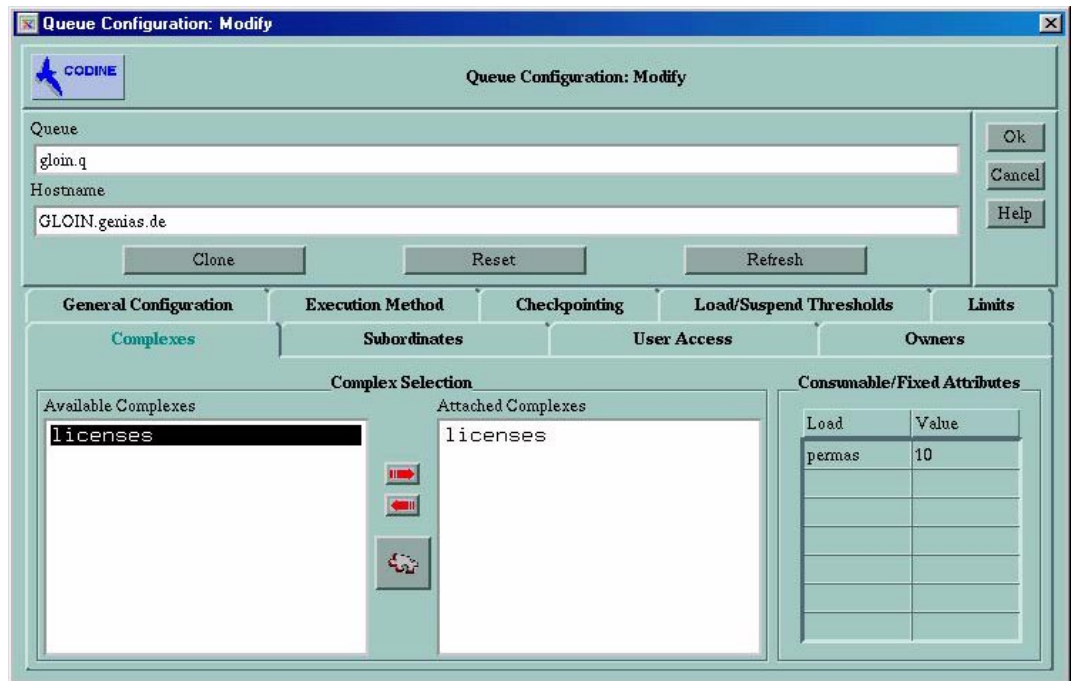


FIGURE 2-19 Queue Configuration “User Complexes”

Configuring Subordinate Queues

If the `Subordinates` parameter set is selected, the parameter set definition region looks as displayed in figure 2-20 below. The fields offered allow for setting the following parameters:

- The queues which are *subordinated* to the configured queue. Subordinated queues are suspend if the configured queue becomes *busy* and are unsuspended if the configured queue is no longer busy. For any subordinated queue the number of job slots can be configured which at least has to be occupied in the configured queue to trigger a suspension. If no job slot value is specified, all slots need to be filled to trigger suspension of the corresponding queue.

Please refer to the `queue_conf` manual page for detailed information on these parameters.

Use the subordinate queue facility to implement high priority and low priority queues as well as stand-alone queues.

Queue Configuration: Modify

Queue: gloin.q

Hostname: GLOIN.genias.de

Clone Reset Refresh

Ok Cancel Help

General Configuration Execution Method Checkpointing Load/Suspend Thresholds Limits

Complexes Subordinates User Access Owners

| Queue | Max Slots |
|-----------|-----------|
| fangorn.q | 2 |
| | |
| | |
| | |
| | |
| | |

If 'Max Slots' are filled in the current queue, the queues in column one are suspended.

FIGURE 2-20 Queue Configuration “Subordinates”

Configuring User Access

If the `User Access` parameter set is selected, the parameter set definition region looks as displayed in figure 2-21 below. The fields offered allow for setting the following parameters:

- The user access lists being attached to the allow or deny lists of the queue. Users or user groups belonging to access lists which are included in the allow list have access to the queue. Those being associated with the deny list may not access the queue. If the allow list is empty access is unrestricted unless explicitly stated otherwise in the deny list.

Please refer to the `queue_conf` manual page for detailed information on these parameters.

The `Access List Configuration` dialogue (see “User Access Permissions” on page 122) is opened upon clicking on the icon button in the middle bottom of the screen.

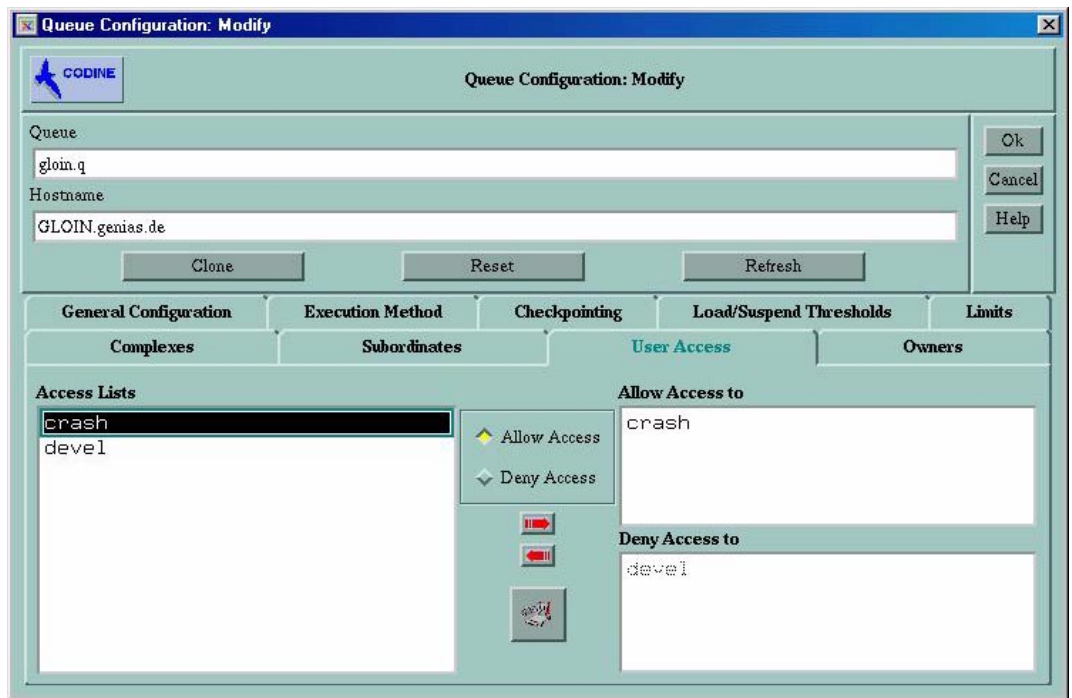


FIGURE 2-21 Queue Configuration “User Access” parameters

Configuring Owners

If the `Owners` parameter set is selected, the parameter set definition region looks as displayed in figure 2-22 below. The fields offered allow for setting the following parameters:

- The list of queue owners. An owner of a queue is given permission to suspend/unsuspend or disable/enable the queue. All feasible user accounts are valid values to be added to the queue owner list. To delete an user account from the queue owner list select it in the `Owner List` window and click on the garbage bin icon in the right lower corner of the dialogue.

Please refer to the `queue_conf` manual page for detailed information on these parameters.

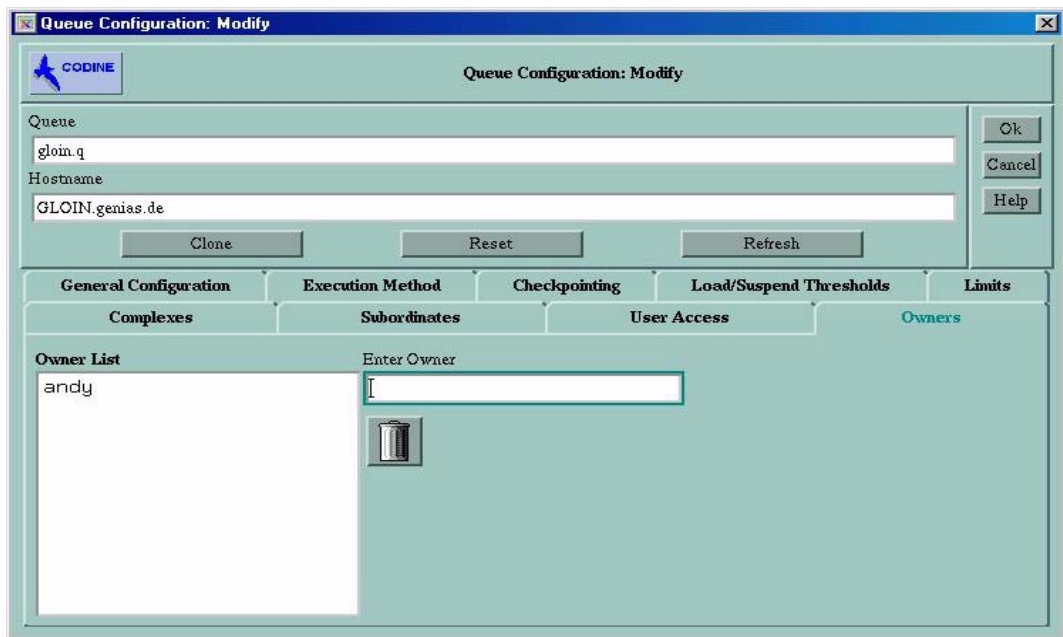


FIGURE 2-22 Queue Configuration “Owners”

Configuring Queues from the Command-line

The queue configuration is maintained by the following `qconf` command options:

`qconf -aq [queue_name]`

add queue. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with a queue configuration template. If the optional parameter `queue_name` is present the configuration of this queue is used as template. The queue is configured by changing the template and saving to disk. See the `queue_conf` manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.

`qconf -Aq file_name`

add queue. Uses the file `file_name` to define a queue. The definition file might have been produced by `qconf -sq queue_name` (see below).

`qconf -cq queue_name[,...]`

clean queue. Cleans the status of the specified queue(s) to be idle and free from running jobs. The status is reset without respect to the current status. The option is useful for eliminating error conditions, but should not be used in normal operation mode.

`qconf -dq queue_name[,...]`

delete queue. Deletes the queue(s) specified in the argument list from the list of available queues.

`qconf -mq queue_name`

modify queue. Modifies the specified queue. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with the configuration of the queue to be changed. The queue is modified by changing the configuration and saving to disk.

`qconf -Mq file_name`

modify queue. Uses the file `file_name` to define the modified queue configuration. The definition file might have been produced by `qconf -sq queue_name` (see below) and subsequent modification.

`qconf -sq [queue_name[,...]]`

show queue. Either displays the default template queue configuration (if no arguments are present) or the current configuration of the queues enlisted in the comma separated argument list.

`qconf -sql`

show queue list. Displays a list of all currently configured queues.

The Complexes Concept

The definition of complexes provides all pertinent information concerning the resource attributes a user may request for a Sun Grid Engine job via the `qsub` or `qalter -l` option and for the interpretation of these parameters within the Sun Grid Engine system.

Complexes also build the framework for Sun Grid Engine's so called *Consumable Resources* facility, a feature allowing for the definition of cluster global, host specific or queue related attributes which identify a resource with an associated capacity. Availability of resources in combination with the requirements of Sun Grid Engine jobs will be taken into account during the scheduling process. Sun Grid Engine will also perform the bookkeeping and capacity planning required to prevent from oversubscription of consumable resources. Examples for typical consumable attributes are available free memory, unoccupied licenses of a software package, free disk space or available bandwidth on a network connection.

In a more general sense, Sun Grid Engine complexes are used as a means for describing the intended interpretation of queue, host and cluster attributes. The description includes the attribute name, a shortcut which can be used to reference it, the value type (e.g. `STRING` or `TIME`) of an attribute, a pre-defined value being assigned to the complex attribute, a relation operator used by the Sun Grid Engine scheduler `cod_schedd`, a requestable flag which determines whether the attribute may be requested for a job by a user or not, a consumable flag which identifies the attribute as consumable attribute if set and a default request value taken into account for consumable attributes if jobs do not explicitly specify their request for such an attribute.

The `qmon` complex configuration dialogue shown below illustrates how complex attributes can be defined. It can be opened either by pushing the `Complex Configuration` icon button in the `qmon` main menu or the corresponding icon button in the `User Complexes` queue and host configuration sub-dialogues. It provides the means for changing the definition of the existing complexes and for defining new user complexes.

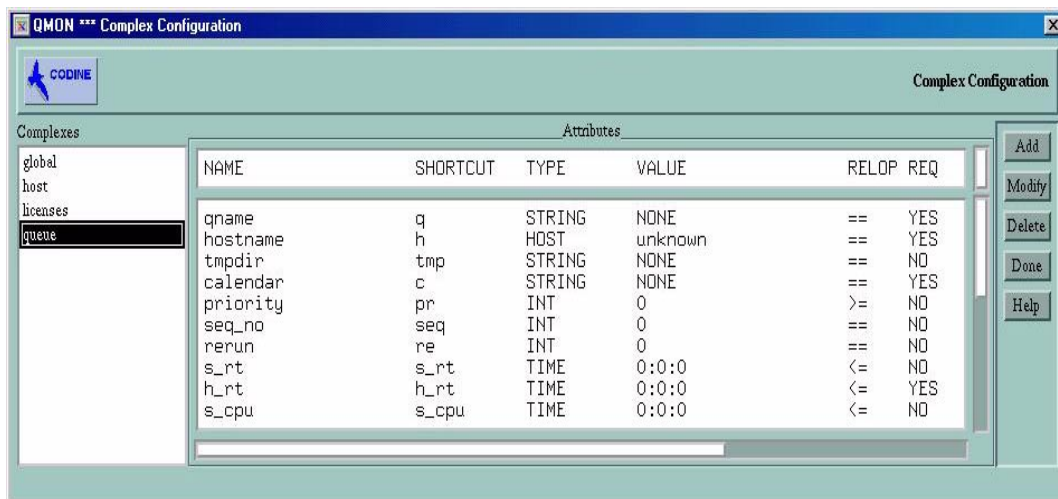


FIGURE 2-23 Complex Configuration dialogue “queue”

On the left side of the screen a selection list for all complexes known to the system is displayed. It can be used if a complex is to be modified or deleted. The desired operation (Add, Modify or Delete) can be selected with the corresponding buttons on the right side of the screen. If a new complex is to be created or an existing complex is modified, the following dialogue is opened.

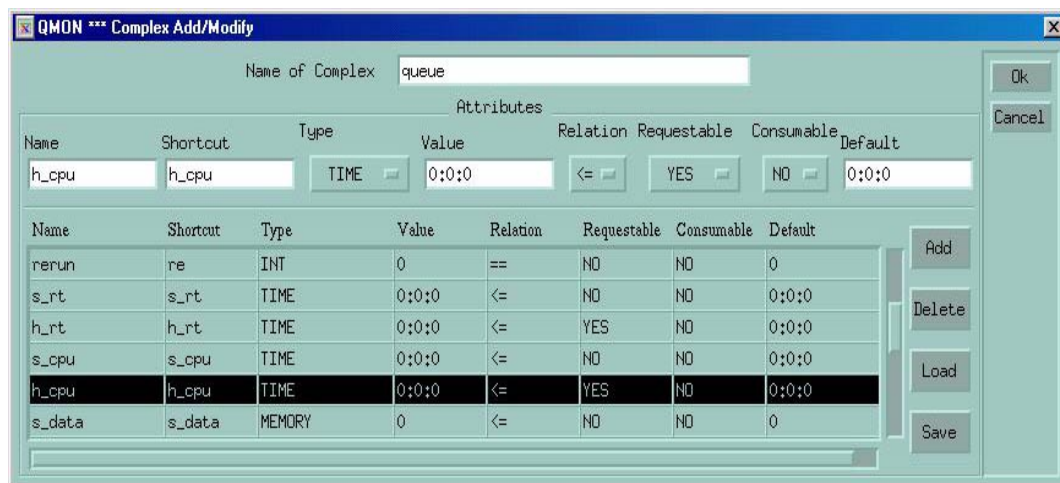


FIGURE 2-24 Complex Add/Modify dialogue

The name of the complex has to be entered or is displayed in the Name of Complex input window at the top. The complex attributes can be modified in the complex definition table by selecting a line with the left mouse button. The selected entry will be displayed in the definition windows and selectors at the top of the Attributes box. Changing the definition and pressing the Add button will update the changes in the definition table.

A new entry can be added by filling out the definition windows and using the selectors and then pressing the Add button. No line in the attributes table should be selected when adding new items.

The Load and Save buttons can be used to load and save complex configurations from and to regular files. A file selection box is opened to select the files. The Delete button can be used to delete selected lines in a complex configuration.

Please refer to the complex manual page for details on the meaning of the rows and columns in the table. The Ok button in the upper right corner of the screen will finally register the new/changed complex with `cod_qmaster`.

Complex Types

The Sun Grid Engine complexes object integrates four different types of complexes:

The *Queue* Complex

It is referenced by the special name `queue`.

In its default form it contains a selection of parameters in the queue configuration as defined in `queue_conf`. The main purpose of the queue complex is to define how these parameters are to be interpreted and to provide a container for further attributes which are intended to be available for all queues. The queue complex thus can be extended by user defined attributes.

If the queue complex is referenced in context with a particular queue, the corresponding configuration values of the queue replace the attribute values (they *overwrite* the `value` column) in the queue complex.

If, for example, the queue complex is setup for a queue called *big*, the value column for the queue complex attribute `qname`, which carries the default value `unknown` (see figure 2-23 on page 89), is set to `big`.

This implicit value setting can be overwritten by using the `complex_values` parameter in the queue configuration (see section “Configuring Queues” on page 75). This is usually done for so called *Consumable Resources* (see section “Consumable Resources” on page 96). For the virtual memory size limit, for example, the queue configuration value `h_vmem` would be used to limit the amount of total occupied

memory per job, while a corresponding entry in the `complex_values` list would define the total available amount of virtual memory on a host or assigned to a queue.

If the administrator adds attributes to the queue complex, their value in association with a particular queue is either defined via the `complex_values` parameter of that queue or the `value` column in the queue complex configuration is used by default.

The *Host* Complex

It is referenced by the special name `host` and contains the characteristics definition of all attributes which are intended to be managed on a host basis (figure 2-25 on page 92). The standard set of host related attributes consists of two categories, but it may be enhanced likewise the queue complex described above. The first category is built by several queue configuration attributes which are particularly suitable to be managed on a host basis. These attributes are:

- `slots`
- `seven`
- `h_vmem`
- `s_fsize`
- `h_fsize`

(please refer to `queue_conf` for details).

Note – Defining these attributes in the host complex is no contradiction to having them also in the queue configuration. It allows maintaining the corresponding resources on a host level and at the same time on a queue level. Total virtual free memory (`h_vmem`) can be managed for a host, for example, and a subset of the total amount can be associated with a queue on that host.

The second attribute category in the standard host complex are the default load values. Every `cod_execd` periodically reports load to `cod_qmaster`. The reported load values are either the standard Sun Grid Engine load values such as the CPU load average or load values defined by the Sun Grid Engine administration (see section “Load Parameters” on page 113). The characteristics definition for the standard load values is part of the default host complex, while administrator defined load values require extension of the host complex.

The host complex commonly is not only extended to include non-standard load parameters, but also to manage host related resources such as the number of software licenses being assigned to a host or the available disk space on a host local filesystem.

If the host complex is associated with a host or a queue on that host, a concrete value for a particular host complex attribute is determined by either

- the queue configuration in the case of the queue configuration derived attributes,
- a reported load value or
- the explicit definition of a value in the `complex_values` entry of the corresponding host configuration (see section “Configuring Hosts” on page 57).

If none of the above is available (e.g. the value is supposed to be a load parameter, but `cod_execd` does not report a load value for it), the `value` field in the host complex configuration is used.

The total free virtual memory attribute `h_vmem`, for example, is defined in the queue configuration as `limit` and is also reported as a standard load parameter. The total available amount of virtual memory on a host and attached to a queue on that host may be defined in the `complex_values` lists of that host and that queue configuration. Together with defining `h_vmem` as *consumable resource* (see section “Consumable Resources” on page 96) this allows to efficiently exploit memory of a machine without risking memory oversubscription often resulting in reduced system performance caused by *swapping*.

Note – Only the `Shortcut`, `Value`, `Relation`, `Requestable`, `Consumable` and `Default` columns may be changed for the system default load attributes. No default attributes should be deleted.

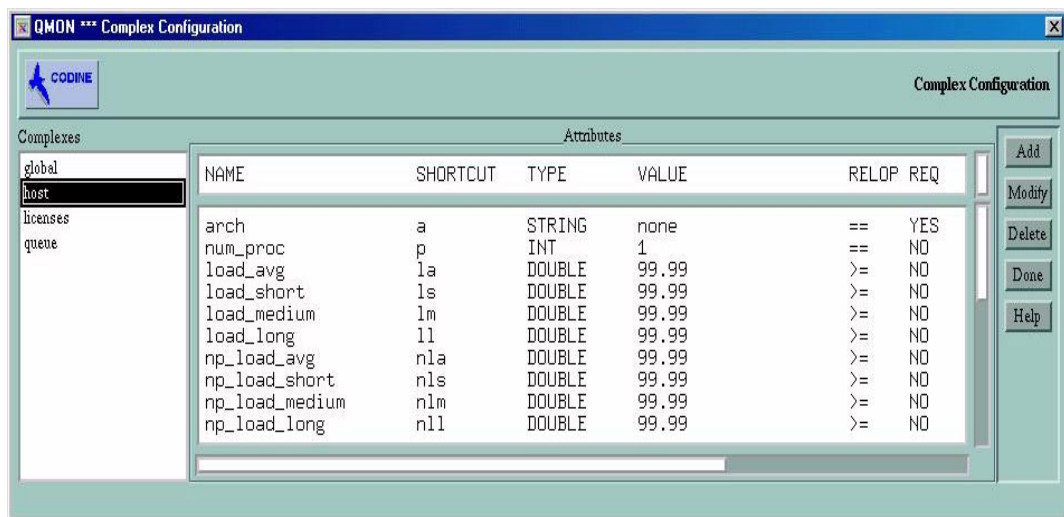


FIGURE 2-25 Complex Configuration dialogue “host”

The *Global* Complex:

It is referenced by the special complex name `global`.

The entries configured in the global complex refer to cluster wide resource attributes, such as available network bandwidth of a file server or the free disk space on a network wide available filesystem (figure 2-26 on page 93). Global resource attributes can also be associated with load reports, if the corresponding load report contains the `GLOBAL` identifier (see section “Load Parameters” on page 113). Global load values can be reported from any host in the cluster. There are no global load values reported by Sun Grid Engine by default and hence there is no default global complex configuration.

Concrete values for global complex attributes are either determined by global load reports, by explicit definition in the `complex_values` parameter of the `global` host configuration (see section “Configuring Hosts” on page 57) or in association with a particular host or queue and an explicit definition the corresponding `complex_values` lists. If none of the above is the case (e.g. a load value has not yet been reported), the `value` field in the global complex configuration is used.

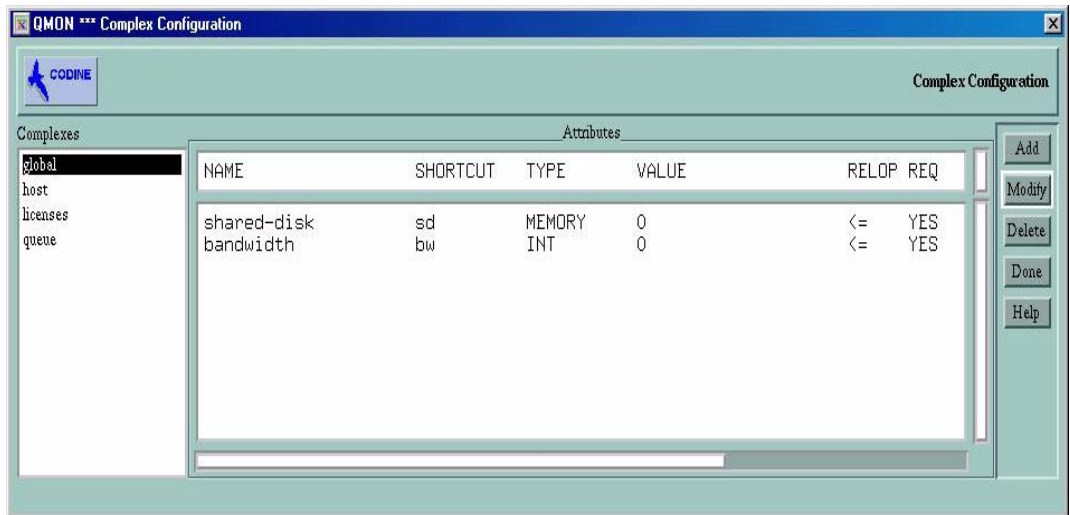


FIGURE 2-26 Complex Configuration dialogue “global”

User Defined Complexes

By setting up user defined complexes the Sun Grid Engine administration has the ability to extend the set of attributes managed by Sun Grid Engine while restricting the influence of those attributes to particular queues and/or hosts. A user complex is just a named collection of attributes and the corresponding definition as to how

these attributes are to be handled by Sun Grid Engine. One or more of these user defined complexes can be attached to a queue and/or host via the `complex_list` queue and host configuration parameter (see section "Configuring Queues" on page 75 and "Configuring Hosts" on page 57). The attributes defined in all assigned complexes become available to the queue and the host respectively in addition to the default complex attributes.

Concrete values for user defined complexes in association with queues and hosts have to be set by the `complex_values` parameter in the queue and host configuration or otherwise the `value` field of the user complex configuration is used.

As an example let the following user defined complex licenses be defined:

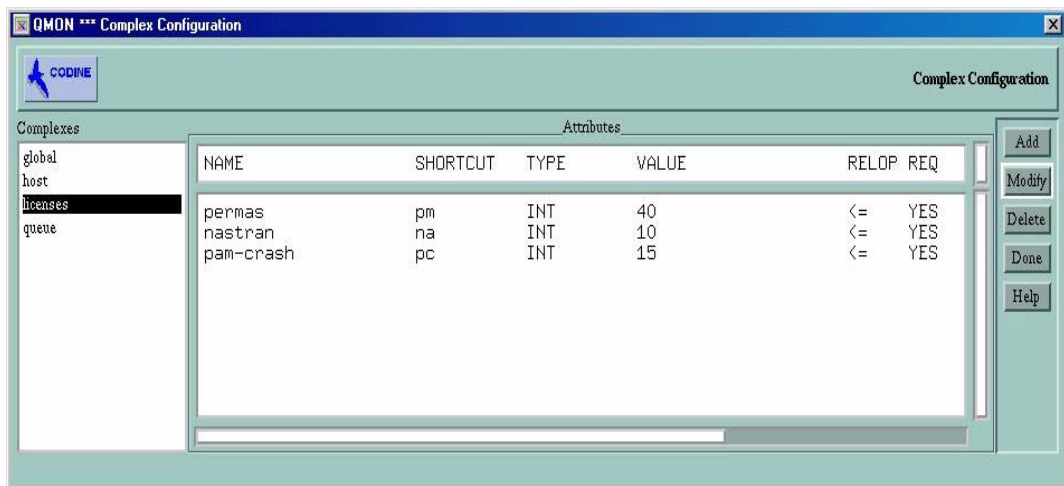


FIGURE 2-27 Complex Configuration dialogue "licenses"

And let for at least one or multiple queues the `licenses` complex be added to the list of associated user defined complexes as show in the queue configuration User Complexes sub-dialogue displayed below (please see section "Configuring Queues" on page 75 for details on how to configure queues):

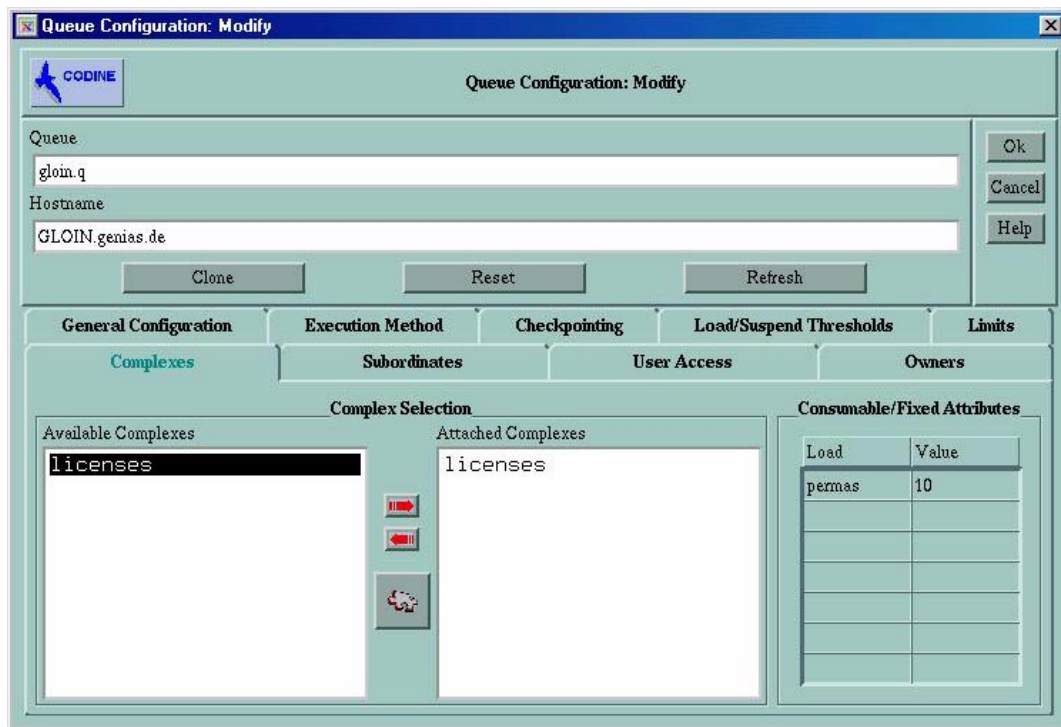


FIGURE 2-28 User Complexes “queue” Configuration

Then the displayed queue is configured to manage up to 10 licenses of the software package `permas`. Furthermore, the `licenses` complex attribute `permas` becomes requestable for Sun Grid Engine jobs as expressed in the Available Resources list in the Requested Resources sub-dialogue of the submit dialogue shown below (see section “Submitting Sun Grid Engine Jobs” on page 175 of the *Sun Grid Engine User’s Guide* for details on how to submit jobs).



FIGURE 2-29 Requested Resources submit sub-dialogue

Alternatively the user could submit jobs from the command-line and request licenses attributes as follows:

```
% qsub -l pe=1 permas.sh
```

Note – The `pm` shortcut could have been used instead of the full attribute name `permas`.

As a consequence of such a configuration and similar job requests, the only queues being eligible for these jobs would be the ones which are associated with the user defined licenses complex, which have `permas` licenses configured and available.

Consumable Resources

Consumable resources, also called *consumables*, are an efficient means to manage limited resources such as available memory, free space on a file system, network bandwidth or floating software licenses. The total available capacity of a consumable is defined by the Sun Grid Engine administrator and the consumption of the corresponding resource is monitored by Sun Grid Engine internal bookkeeping. Sun

Grid Engine accounts for the consumption of this resource for all running jobs and ensures that jobs are only dispatched if the Sun Grid Engine internal bookkeeping indicates enough available consumable resources.

Consumables can be combined with default or user defined load parameters (see section "Load Parameters" on page 113), i.e. load values can be reported for consumable attributes or conversely the `Consumable` flag can be set for load attributes. The Sun Grid Engine consumable resource management takes both the load (measuring availability of the resource) and the internal bookkeeping into account in this case, and makes sure that neither of both exceeds a given limit.

To enable consumable resource management the total capacity of a resource has to be defined. This can be done on a cluster global, per host and per queue basis while these categories may supersede each other in the given order (i.e. a host can restrict availability of a cluster resource and a queue can restrict host and cluster resources). The definition of resource capacities is performed with the `complex_values` entry in the queue and host configuration (see `host_conf` and `queue_conf` as well as "Configuring Queues" on page 75 and "Configuring Hosts" on page 57). The `complex_values` definition of the `global` host specifies cluster global consumable settings. To each consumable complex attribute in a `complex_values` list a value is assigned which denotes the maximum available amount for that resource. The internal bookkeeping will subtract from this total the assumed resource consumption by all running jobs as expressed through the jobs' resource requests.

Setting Up Consumable Resources

Only numeric complex attributes (those with type `INT`, `MEMORY` and `TIME`) can be configured as consumables. To switch on the Sun Grid Engine consumable management for an attribute, you first have to set the `CONSUMABLE` flag for it in the complex configuration as depicted in "Complex Configuration dialogue `"virtual_free"`" on page 98 for the `virtual_free` memory resource.

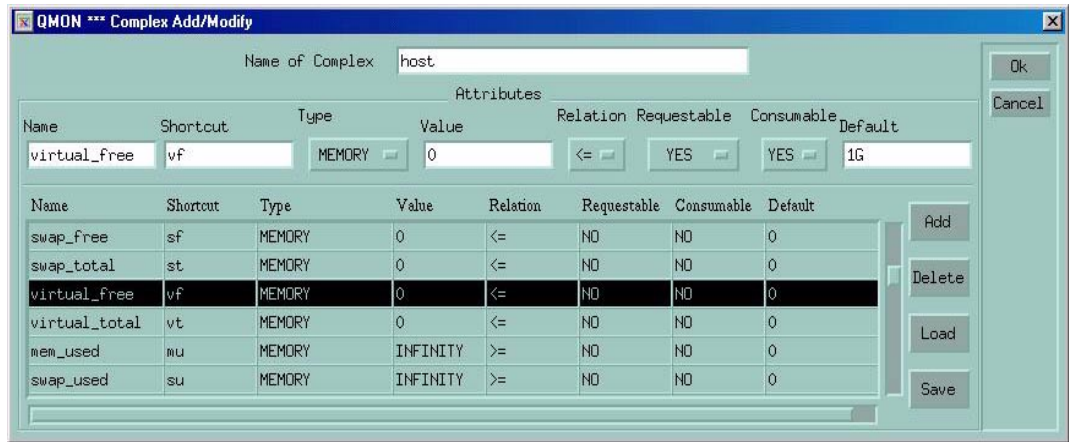


FIGURE 2-30 Complex Configuration dialogue “virtual_free”

Then, for each queue or for each host you want Sun Grid Engine to do the required capacity planning, you have to define the capacity in a `complex_values` list. An example is shown in figure "Execution Host Configuration “virtual_free”" on page 99 where 1 Gigabyte of virtual memory is defined as capacity value of the current host.

The virtual memory requirements of all jobs running concurrently on that host (in any queue) will be accumulated and subtracted from the capacity of 1 Gigabyte to determine available virtual memory. If a job request for `virtual_free` exceeds the available amount, the job will not be dispatched to a queue on that host.

Note – Jobs can be forced to request a resource and thus to specify their assumed consumption via the *force* value of the Requestable parameter (see figure 2-30 on page 98).

Note – A default resource consumption value can be pre-defined by the administrator for consumable attributes not explicitly requested by the job (see figure 2-30 on page 98 - 200 Megabytes are set as default). This is meaningful only if requesting the attribute is not enforced as explained above.

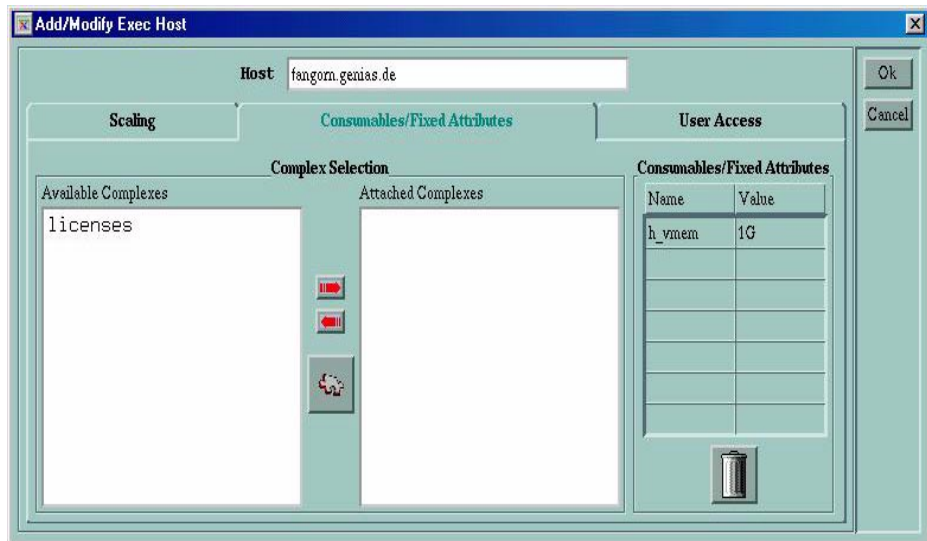


FIGURE 2-31 Execution Host Configuration “virtual_free”

Examples

Example 1: Floating Software License Management

Suppose you have the software package PAM-CRASH in use in your cluster and you have access to 10 floating licenses, i.e. you can use PAM-CRASH on every system as long as the total active invocations of the software do not exceed the number 10. The goal is to configure Sun Grid Engine in a way which prevents from scheduling PAM-CRASH jobs as long as all 10 licenses are occupied by other running PAM-CRASH jobs.

With Sun Grid Engine consumable resources this can be achieved easily! First, you need to add the number of available PAM-CRASH licenses as a consumable resource to the global complex configuration as shown in figure 2-32 on page 100.

QMON *** Complex Add/Modify

Name of Complex:

Attributes

| Name | Shortcut | Type | Value | Relation | Requestable | Consumable | Default |
|-----------|----------|------|-------|----------|-------------|------------|---------|
| pam-crash | pc | INT | 0 | <= | FORCED | YES | 1 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Buttons: Add, Delete, Load, Save, Ok, Cancel

FIGURE 2-32 Complex Configuration dialogue “pam-crash”

The name of the consumable attribute is set to *pam-crash* and *pc* can be used as short-cut in the *qalter*, *qselect*, *qsh*, *qstat* or *qsub -l* option instead. The attribute type is defined to be an integer counter. The setting of the Value field is irrelevant for consumable resources as they receive their value from the global, host or queue configurations via the *complex_values* lists (see below). The Requestable flag is set to FORCED to indicate that users have to request how much PAM-CRASH licenses their job will occupy when submitting it. The Consumable flag finally defines the attribute to be a consumable resource while the setting of Default is irrelevant since Requestable is set to FORCED and thus a request value will be received for this attribute with any job.

To activate resource planning for this attribute and for the cluster the number of available PAM-CRASH licenses has to be defined in the *global* host configuration as displayed in figure 2-33 on page 101. The value for the attribute *pam-crash* is set to 10 corresponding to 10 floating licenses.

Note – The table Consumable/Fixed Attributes corresponds to the *complex_values* entry described in the host configuration file format *host_conf*.

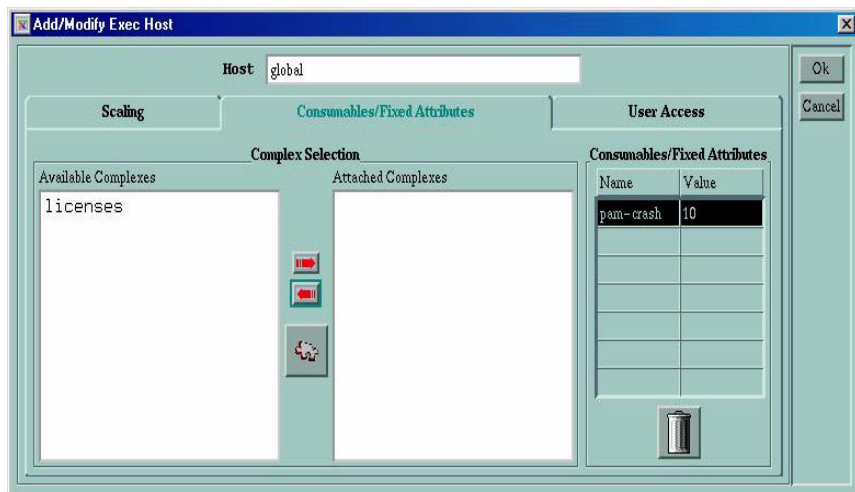


FIGURE 2-33 Global Host Configuration “pam-crash”

If a user now submits the following job:

```
% qsub -l pc=1 pam-crash.sh
```

it will only get started if less than 10 PAM-CRASH licenses are currently occupied. The job may run anywhere in the cluster, however, and it will occupy one PAM-CRASH license for itself throughout its run time.

If one of your hosts in the cluster cannot be included in the floating license, e.g. because you do not have PAM-CRASH binaries for it, you can simply exclude it from the PAM-CRASH license management by setting the capacity related to this host for the consumable attribute `pam-crash` to 0. This has to be done in the execution host configuration dialogue as shown for host `fangorn` in figure 2-34 on page 102.

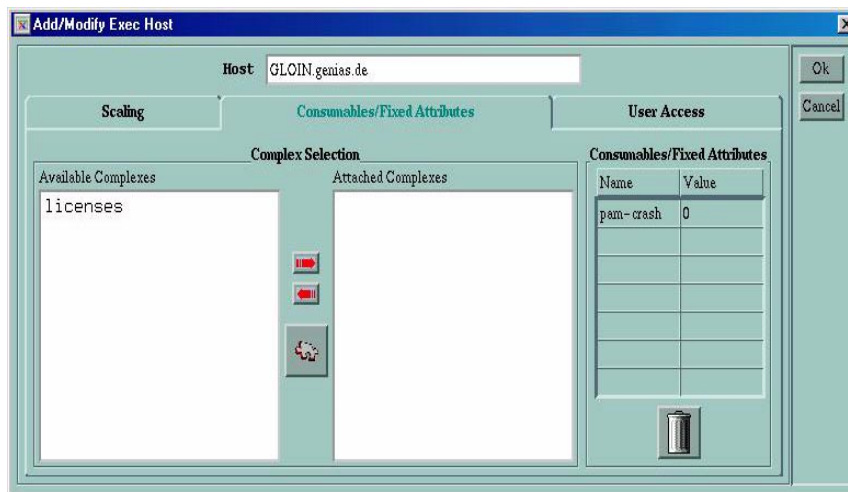


FIGURE 2-34 Execution Host configuration “pam-crash”

Note – The `pam-crash` attribute is implicitly available to the execution host, because the attributes of the `global` complex are inherited to all execution hosts.

Note – Likewise setting the capacity to 0 you could also restrict the number of licenses to be managed by a particular host as part of all licenses of the cluster to a certain non-zero value, such as 2. In this case a maximum of 2 PAM-CRASH jobs could co-exist on that host.

Similarly, you could want to prevent a certain queue from executing PAM-CRASH jobs, e.g. because it is an express queue with memory and CPU-time limits not suitable for PAM-CRASH. In this case you just would have to set the corresponding capacity to 0 in the queue configuration as shown in figure 2-35 on page 103.

Note – The `pam-crash` attribute is implicitly available to the queue, because the attributes of the `global` complex are inherited to all queues.

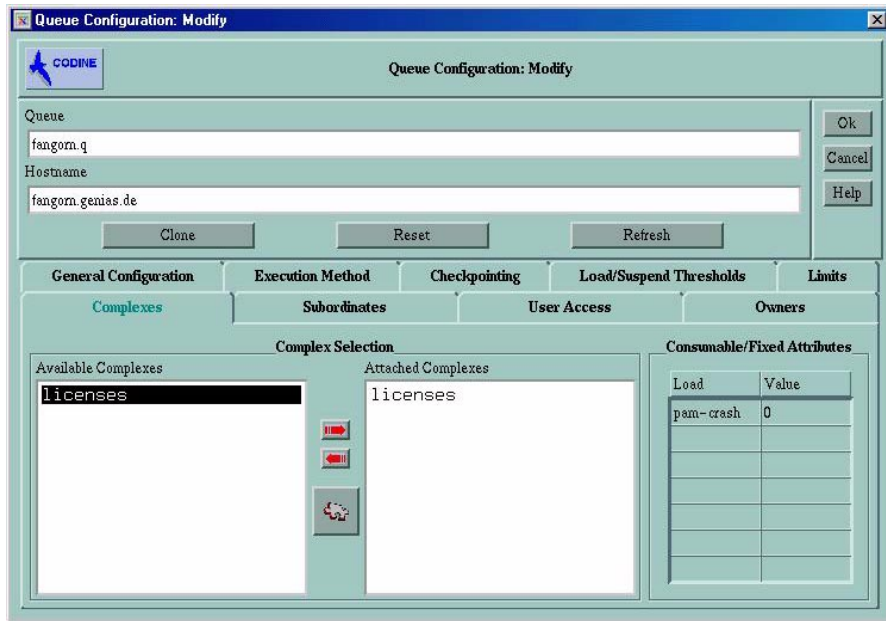


FIGURE 2-35 Queue Configuration “pam-crash”

Example 2: Space Sharing for Virtual Memory

To tune a system in a way that performance degradation caused by memory oversubscription and consequently swapping of a machine is avoided is a common task for system administrators. Sun Grid Engine can support you in this tasks via the consumable resources facility.

The standard load parameter `virtual_free` is designated to report the available free virtual memory, i.e. the combination of available swap space and the available physical memory. To avoid swapping, the use of swap space has to be minimized, i.e. in an ideal case all the memory required by all processes executing on a host should fit into physical memory.

Sun Grid Engine can guarantee this for all jobs started via Sun Grid Engine given the following assumptions and configurations:

- `virtual_free` is configured as consumable resource and its capacity on each host is set to the available physical memory (or lower).
- Jobs request their anticipated memory usage and the value requested is not exceeded during run time.

An example for a possible host complex configuration is shown in figure "Complex Configuration dialogue "virtual_free"" on page 98 and a corresponding execution host configuration for a host with 1 Gigabyte of main memory is depicted in figure "Execution Host Configuration "virtual_free"" on page 99.

Note – The Requestable flag is set to YES in the host configuration example as opposed to FORCED in the previous example of a global complex configuration. This means, that users do not have to indicate the memory requirements of their jobs, but that the value in the Default field is used if an explicit memory request is missing. The value of 1 Gigabyte as default request in this case means, that a job without request is assumed to occupy all the available physical memory.

Note – virtual_free is one of the standard load parameters of Sun Grid Engine. The additional availability of recent memory statistics will be taken into account automatically by Sun Grid Engine in the virtual memory capacity planning. If the load report for free virtual memory falls below the value obtained by Sun Grid Engine-internal bookkeeping, the load value will be used to avoid memory oversubscription. Differences in the reported load values and the Sun Grid Engine internal bookkeeping may occur easily if jobs are started without using Sun Grid Engine.

If you run a mix of different job classes with typical different memory requirements on a single machine you might wish to partition the memory of the machine for use through these job classes. This functionality, frequently called *space sharing*, can be accomplished by configuring a queue for each job class and by assigning to it a portion of the total memory on that host.

In our example, the queue configuration shown in figure figure 2-36 on page 105 would attach half of the total memory available to host fangorn, i.e. 500 Megabytes, to the queue big_f. Hence the accumulated memory consumption of all jobs executing in queue big_f may not exceed 500 Megabytes. Jobs in other queues are not taken into account, but the total memory consumption of all running jobs on host fangorn may still not exceed 1 Gigabyte.

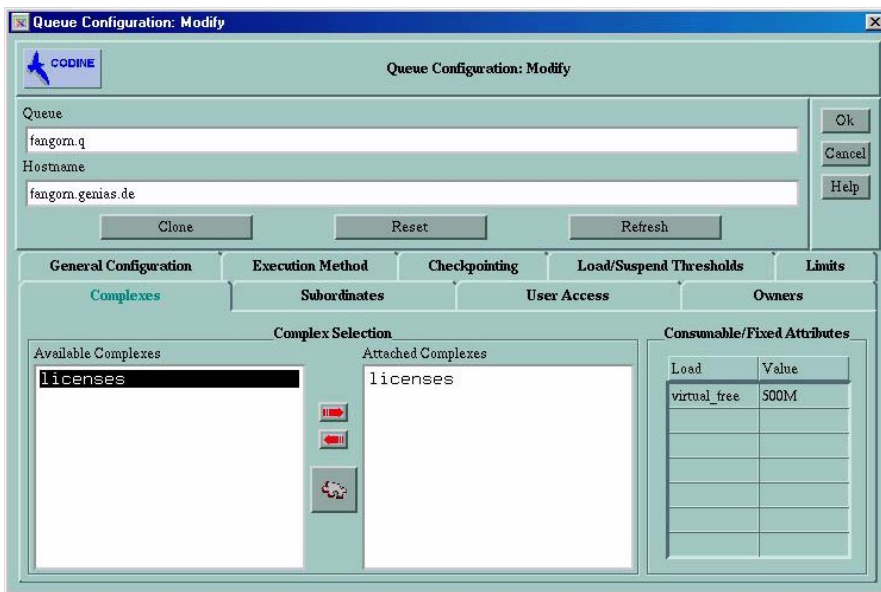


FIGURE 2-36 Queue Configuration “virtual_free”

Note – The attribute `virtual_free` is available to all queues via inheritance from the host complex.

Users might submit jobs to a system configured like in our example case in either of the following forms:

```
% qsub -l vf=100M honest.sh
% qsub dont_care.sh
```

The job submitted by the first command can be started as soon as at least 100 Megabytes of memory are available and this amount will be taken into account in the capacity planning for the `virtual_free` consumable resource. The second job will only run if no other job is on the system as it implicitly request all the available memory. In addition, it will not be able to run in queue `big_f` because it exceeds the queue’s memory capacity.

Example 3: Managing Available Disk Space

Some applications need to manipulate huge data sets stored in files and hence depend on availability of sufficient disk space throughout their run time. This requirement is similar to the space sharing of available memory as discussed in the preceding example. The main difference is that Sun Grid Engine does not provide free disk space as one of its standard load parameters. This is due to the fact that disks are usually partitioned into file systems in a site specific way, which does not allow to identify the file system *of interest* automatically.

Nevertheless, available disk space can be managed efficiently by Sun Grid Engine via the consumables resources facility. It is recommended to use the host complex attribute `h_fsize` for this purpose for reasons explained later in this section. First, the attribute has to be configured as consumable resource, for instance as shown in figure 2-37 on page 106.

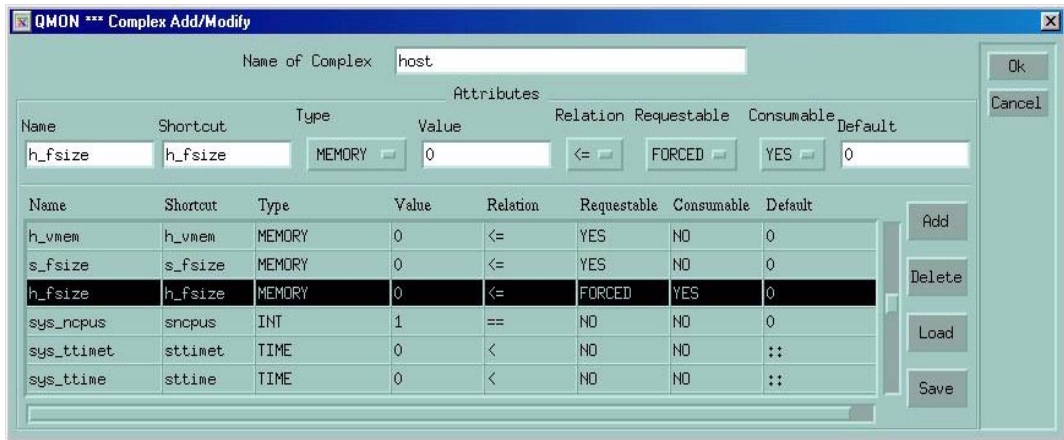


FIGURE 2-37 Complex Configuration dialogue “h_fsize”

If we assume host local file systems, it is reasonable to put the capacity definition for the disk space consumable to the host configuration as shown in figure 2-38 on page 107.

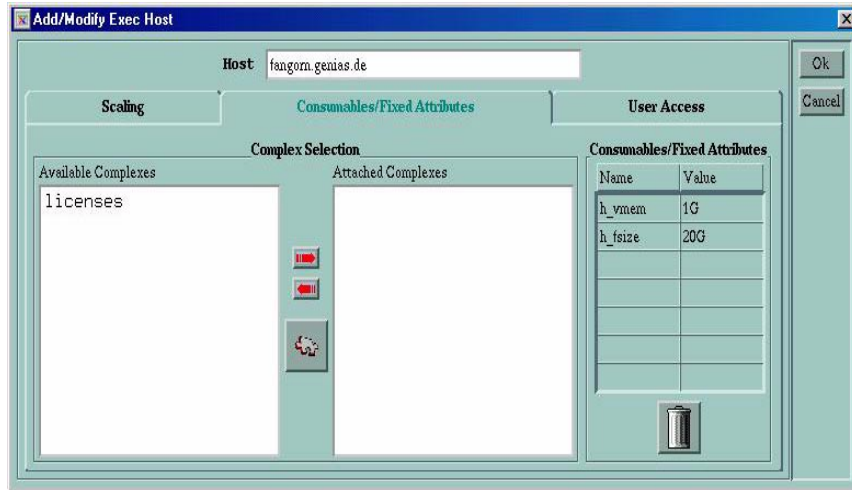


FIGURE 2-38 Execution Host configuration “h_fsize”

Submitting jobs to a Sun Grid Engine system configured in such a way works analogously to the previous examples:

```
% qsub -l hf=5G big_sort.sh
```

The reason why the `h_fsize` attribute has been recommended in this example lies in the fact that `h_fsize` also is used as the *hard file size limit* in the queue configuration. The file size limit is used to restrict the ability of the jobs to create files larger than specified during job submission (5 Gigabyte in the example above) or the corresponding value from the queue configuration if the job does not request the attribute. The Requestable flag for `h_fsize` has been set to FORCED in our example, so a request will always be present.

By using the queue limit as the consumable resource, we automatically gain control on the requests as specified by the user versus the real resource consumption by the job scripts. Any violation of the limit will be sanctioned and the job eventually will be aborted (see the `queue_conf` and the `setrlimit` manual pages for details). This way it can be ensured that the resource requests, on which the Sun Grid Engine internal capacity planning is based, are reliable.

Note – Some operating systems only provide per process file size limits. In this case a job might create multiple files with a size up to the limit. On systems which support per job file size limitation, Sun Grid Engine however uses this functionality with the `h_fsize` attribute (see `queue_conf` for further details).

If you expect applications not being submitted to Sun Grid Engine to occupy disk space concurrently, the Sun Grid Engine internal bookkeeping might not be sufficient to prevent from application failure due to lack of disk space. To avoid this problem it would be helpful to receive disk space usage statistics in a periodical fashion, which would indicate total disk space consumption including the one occurring outside Sun Grid Engine.

The Sun Grid Engine load sensor interface (see “Adding Site Specific Load Parameters” on page 114) allows you to enhance the set of standard Sun Grid Engine load parameters with site specific information, such as the available disk space on a particular filesystem.

By adding an appropriate load sensor and reporting free disk space for `h_fsize` you can combine consumable resource management and resource availability statistics. Sun Grid Engine will compare job requirements for disk space with the available capacity derived from the Sun Grid Engine internal resource planning and with the most recent reported load value. Jobs will only get dispatched to a host if both criteria are met.

Configuring Complexes

Sun Grid Engine complexes can either be defined and maintained graphically via the `qmon` Complex Configuration dialogue shown and explained on page 89 and following or can be performed from the command-line via the `qconf` options `-Ac`, `-ac`, `-Mc`, `-mc` and `-sc`. The command:

```
% qconf -sc licenses
```

prints the `nastran` complex (as define in figure 2-27 on page 94) to the standard output stream in the file format as defined in the `complex` section 5 manual page. A sample output is shown for the `licenses` complex below:

| #name | shortcut | type | value | relop | requestable | consumable | default |
|---|----------|------|-------|-------|-------------|------------|---------|
| #----- | | | | | | | |
| nastran | na | INT | 10 | <= | YES | NO | 0 |
| pam-crash | pc | INT | 15 | <= | YES | YES | 1 |
| permas | pm | INT | 40 | <= | FORCED | YES | 1 |
| #---- # start a comment but comments are not saved across edits ----- | | | | | | | |

TABLE 2-3 `qconf -sc` sample output

Please refer to the `complex` manual page for a detailed definition of the format and the valid value field syntax.

While the `qconf -Ac` and `-Mc` options take such a complex configuration file as an argument, the `-ac` and `-mc` options bring up an editor filled in with a template complex configuration or the configuration of an existing complex for modification.

The meaning of the options is defined as follows:

```
qconf -Ac, -ac
```

Add a new complex to the list of available complexes.

```
qconf -Mc, -mc
```

Modify an existing complex.

Queue Calendars

Queue calendars define the availability of Sun Grid Engine queues dependent on the day of the year, the day of the week and/or the day time. Queues can be configured to change their status at arbitrary points in time. The queue status can be changed to disabled, enabled, suspended and resumed (unsuspended).

Sun Grid Engine provides the ability to define a site specific set of calendars, each of which contains arbitrary status changes and the time events at which they occur. These calendars can be referred to by queues, i.e. each queue may (or may not) attach a single calendar thereby adopting the availability profile defined in the attached calendar.

The syntax of the calendar format is described in `calendar_conf` in detail. A few examples are given below along with a description of the corresponding administration facilities.

Configuration with `qmon`

The queue calendar configuration dialogue (figure 2-39 on page 110) is opened upon clicking to the `Calendar Config` icon button in the `qmon` main menu. The already available access lists are displayed in the `Calendars` selection list on the left side of the screen. The contents of a calendar configuration is displayed in the display region entitled with `Configuration` if it is selected by clicking on it with the left mouse button in the `Calendars` selection list.

A selected calendar can be deleted by pressing the Delete button on the right side of the screen. Selected calendars can be modified after pushing the Modify button and new access lists can be added after pushing the Add button. In both cases, the calendar definition dialogue displayed in figure 2-40 on page 111 is opened and provides the corresponding means:

The Calendar Name input window either displays the name of the selected calendar in the case of a modify operation or can be used to enter the name of the calendar to be declared. The Year and Week input fields allow the definition of the calendar events using the syntax described in `calendar_conf`.

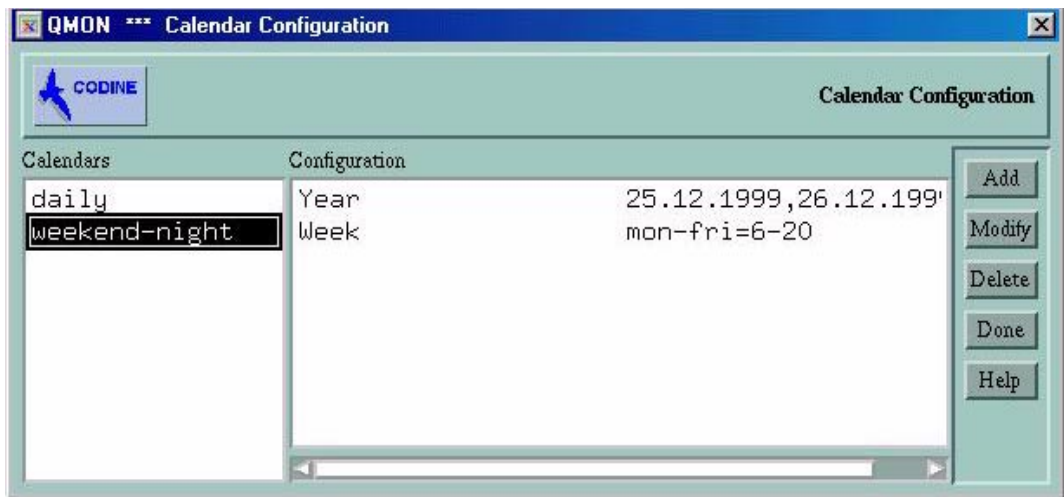


FIGURE 2-39 Calendar Configuration

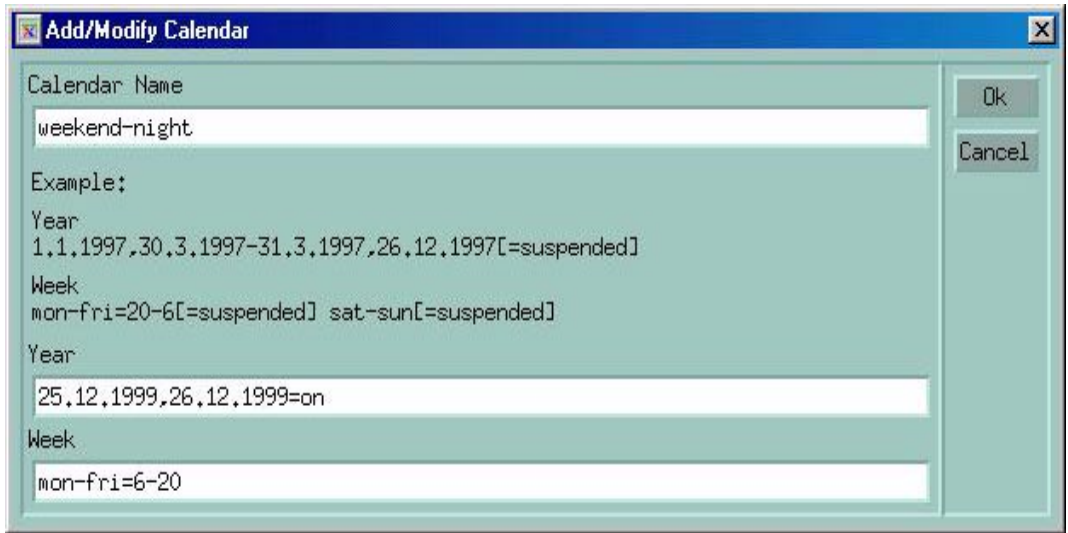


FIGURE 2-40 Add/Modify Calendar

The example calendar configuration above is appropriate for queues which should be available outside office hours and on weekends. In addition, the christmas holidays have been defined to be handled like week ends.

See the `calendar_conf` manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the syntax and for further examples.

By attaching a calendar configuration for a queue the availability profile defined by the calendar is set for the queue. Calendars are attached in the general parameter queue configuration menu as displayed in figure 2-41 on page 112. The Calendar input field contains the calendar name to be attached and the icon button next to the input field opens a selection dialogue with the list of currently configured calendars. See section "Configuring Queues" on page 75 for further detail on configuring queues.

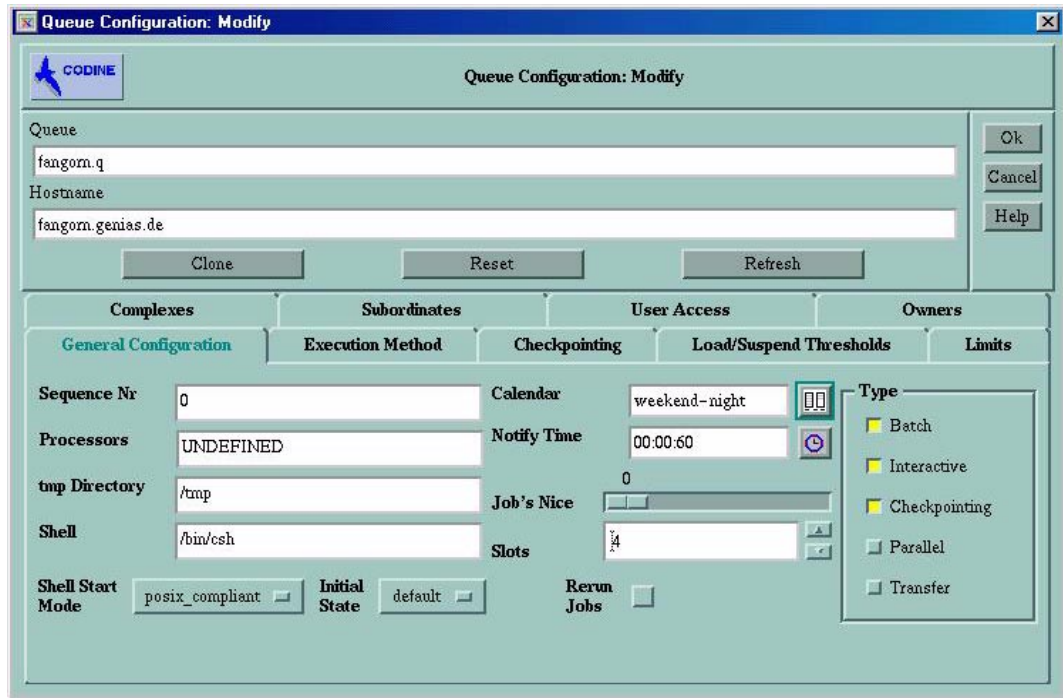


FIGURE 2-41 Queue Configuration “Calendar”

Command-line Configuration

The command-line interface to administer the calendar configuration of your Sun Grid Engine cluster is provided by several switches to the Sun Grid Engine `qconf` command:

```
qconf -Acal, -acal
```

add calendar. Adds a new calendar configuration to the Sun Grid Engine cluster. The calendar to be added is either read from file or an editor with a template configuration is opened to enter the calendar.

```
qconf -dcal
```

delete calendar. Adds a new calendar configuration to the Sun Grid Engine cluster. The calendar to be added is either read from file (`-Acal`) or an editor with a template configuration is opened to enter the calendar (`-acal`).


```
qconf -Mcal, -mcal
```

modify calendar. Modifies an existing calendar configuration. The calendar to be modified is either read from file (-Mcal) or an editor with the previous configuration is opened to enter the new definition (-mcal).

```
qconf -scal, -scall
```

show calendar. Displays an existing calendar configuration (-scal) or prints a list of all configured calendars (-scall).

Load Parameters

The Default Load Parameters

Per default `cod_execd` periodically reports several load parameters and the corresponding values to `cod_qmaster`. They are stored in the `cod_qmaster` internal host object (see section “Sun Grid Engine Daemons and Hosts” on page 56), however, they are used internally only if a complex attribute with a corresponding name is defined. Such complex attributes contain the definition as to how load values have to be interpreted (see section “Complex Types” on page 90 for details).

After the primary installation a standard set of load parameters is reported. All attributes required for the standard load parameters are defined in the host complex. Subsequent releases of Sun Grid Engine may provide extended sets of default load parameters. Therefore, the set of load parameters being reported per default is documented in the file `<codine_root>/doc/load_parameters.asc`.

Note – The complex in which load attributes are defined decides about their accessibility. Defining load parameters in the global complex makes them available for the entire cluster and all hosts. Defining them in the host complex provides the attributes for all hosts but not cluster globally. Defining them in a user defined complex allows to control visibility of the load parameter by attaching or detaching a user complex to a host.

Note – Load attributes should not be defined in queue complexes as they would be neither available to any host nor to the cluster.

Adding Site Specific Load Parameters

The set of default load parameters may not be adequate to completely describe the load situation in a cluster, especial with respect to site specific policies, applications and configurations. Therefore, Sun Grid Engine provides the means to extend the set of load parameters in an arbitrary fashion. For this purpose, `cod_execd` offers an interface to feed load parameters together with the current load values into `cod_execd`. Afterwards, these parameters are treated exactly like the default load parameters. Likewise for the default load parameters (see section “The Default Load Parameters” on page 113) corresponding attributes need to be defined in a load complex for the load parameters to become effective.

How to Write Your Own Load Sensors

In order to feed `cod_execd` with additional load information a so called load sensor has to be supplied. The load sensor may be a script or a binary executable. In either case its handling of the standard input and output stream and its control flow must comply to the following rules:

The load sensor has to be written as infinite loop waiting at a certain point for input from STDIN. If the string **quit** is read from STDIN, the load sensor is supposed to exit. As soon as an end-of-line is read from STDIN a load data retrieval cycle is supposed to start. The load sensor then performs whatever operation is necessary to compute the desired load figures. At the end of the cycle the load sensor writes the result to stdout. The format is as follows:

- A load value report starts with a line containing nothing but the word **begin**.
- Individual load values are separated by newlines.
- Each load value information consists of three parts separated by colons (":") and containing no blanks.
- The first part of a load value information is either the name of the host for which load is reported or the special name `global`.
- The second part is the symbolic name of the load value as defined in the host or global complex list (see `complex(5)` in the *Sun Grid Engine Reference Manual* for details). If a load value is reported for which no entry in the host or global complex list exists, the reported load value is not used.
- The third part is the measured load value.
- A load value report
- ends with a line with the word **end**.

A sample Bourne shell script load sensor may look as follows:

```
#!/bin/sh
myhost=`uname -n`
while [ 1 ]; do
    # wait for input
    read input
    result=$?
    if [ $result != 0 ]; then
        exit 1
    fi
    if [ $input = quit ]; then
        exit 0
    fi
    #send users logged in
    logins=`who | cut -f1 -d" " | sort | uniq | wc -l` | sed "s/^ *//"
    echo begin
    echo "$myhost:logins:$logins"
    echo end
done
# we never get here
exit 0
```

If this example is saved into the file `load.sh` and executable permission is assigned to it via `chmod`, it can be tested interactively from the command-line simply by invoking `load.sh` and pressing repeatedly the **<return>** button of the keyboard.

As soon as the procedure works, it can be installed for any execution host simply by configuring the path of the load sensor as the `load_sensor` parameter for the cluster global or the execution host specific configuration (see section “Cluster Configuration” on page 70 or the `sgc_conf` manual page).

The corresponding qmon screen might look as follows:

Cluster Settings

Configuration for Host

FANGORN.genias.de

General Settings

Master Spool Dir:

Execd Spool Dir:

QSI Common Dir:

Binary Path:

Mailer:

Xterm:

Load Sensor:

Admin User:

Admin Mail:

Prolog:

Epilog:

Login Shells:

Shell Start Mode:

Log Level:

Advanced Settings

Min UID: Min GID: Finished Jobs:

Loadreport Time: Max Unheard:

Stat Log Time:

User Lists:

Xuser Lists:

Ok Cancel

FIGURE 2-42 Local Configuration with load sensor

The reported load parameter logins will be usable, as soon as a corresponding attribute is added to the host complex. The required definition might look as the last table entry in the example qmon Complex Configuration screen below.

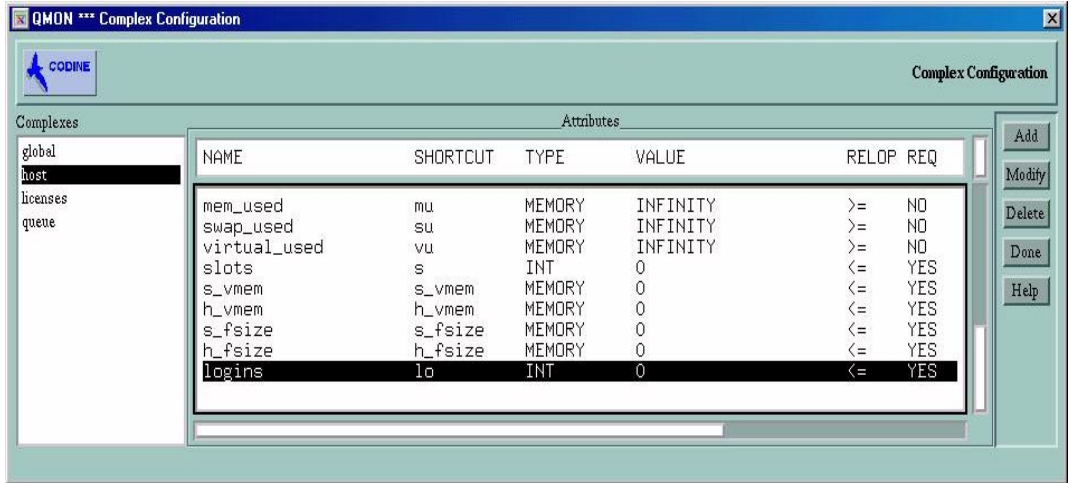


FIGURE 2-43 Complex Configuration dialogue “logins”

Managing User Access

There are four user categories in Sun Grid Engine:

1. Managers:

Managers have full capabilities to manipulate Sun Grid Engine. By default, the superusers of the master host and any machine hosting a queue have manager privileges.

2. Operators:

The operators can perform the same commands as the manager except that they cannot add/delete/modify queues.

3. Owners:

The queue owners are restricted to suspending/unsuspending or disabling/enabling the owned queues. These privileges are necessary for successful usage of `qidle`. Users are commonly declared to be owner of the queues residing on their desk workstation.

4. Users:

Users have certain access permissions as described in "User Access Permissions" on page 122 but no cluster or queue management capabilities.

Each category is described in more detail by the subsequent sections.

Manager Accounts

Configure Manager Accounts with qmon

A set of user access configuration dialogues is invoked by pushing the `User Config` icon button in the `qmon` main menu. The available dialogues are the *Manager Account Configuration* (see figure 2-44), the *Operator Account Configuration* (see figure 2-45) and the *User Access List Configuration* dialogue (see figure 2-46). The dialogues can be switched by using the tab selectors at the top of the screen. The manager account configuration dialogue is opened by default when the `User Config` button is pressed for the first time.

If the `Manager` tab is selected the *Manager Configuration* dialogue (see figure 2-44) is presented and accounts can be declared which are allowed to execute any administrative Sun Grid Engine command. The selection list in the lower half of the screen displays the accounts already declared to have administrative permission. An existing manager account can be deleted from this list by clicking on its name with the left mouse button and by pushing the `Delete` button at the right side of the dialogue. A new manager can be added by entering its name to the input window above the selection list and pressing the `Add` button afterwards or pressing the `Return` key on the keyboard alternatively.

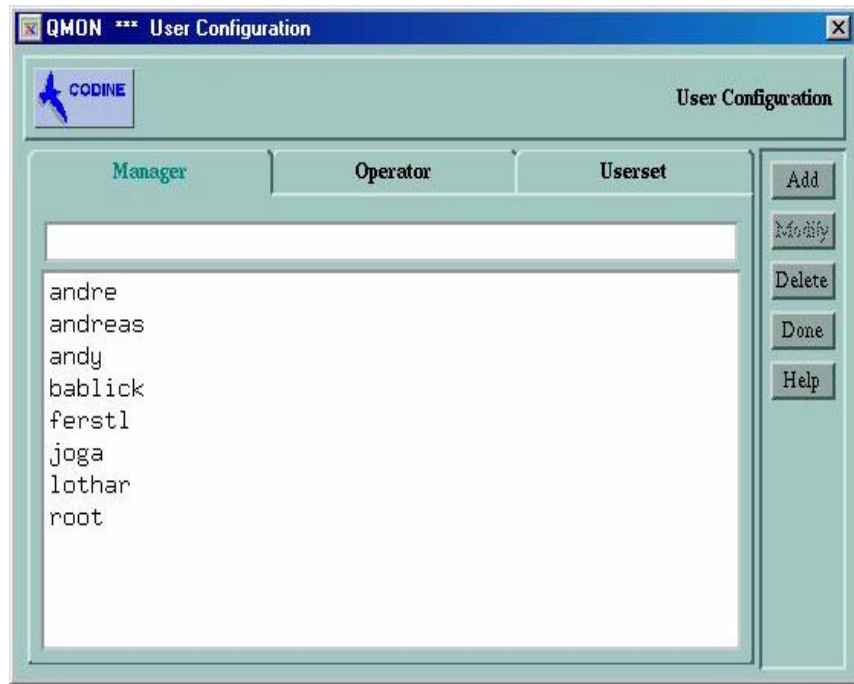


FIGURE 2-44 Manager Configuration dialogue

Configure Manager Accounts from the Command-line

The command-line interface to administer the manager accounts of your Sun Grid Engine cluster is provided by several switches to the Sun Grid Engine `qconf` command:

```
qconf -am user_name[,...]
```

add manager. Adds one or multiple users to the list of Sun Grid Engine managers. By default the root accounts of all Sun Grid Engine trusted hosts (see section "Sun Grid Engine Daemons and Hosts" on page 56) are Sun Grid Engine managers.

```
qconf -dm user_name[,...]
```

delete manager. Deletes the specified user(s) from the list of Sun Grid Engine managers.

```
qconf -sm
```

show managers. Show the list of all Sun Grid Engine managers.

Operator Accounts

Configure Operator Accounts with qmon

The Operator Configuration dialogue (see figure 2-45) is opened upon pushing the User Config button in the qmon main menu and selecting the Operator tab. Accounts can be declared which have restricted administrative Sun Grid Engine command permission unless they are declared to be manager accounts also (see “Manager Accounts” on page 118). The selection list in the lower half of the screen displays the accounts already declared to provide operator permission. An existing account can be deleted from this list by clicking on its name with the left mouse button and by pushing the Delete button at the right side of the dialogue. A new operator can be added by entering its name to the input window above the selection list and pressing the Add button afterwards or pressing the Return key on the keyboard alternatively.

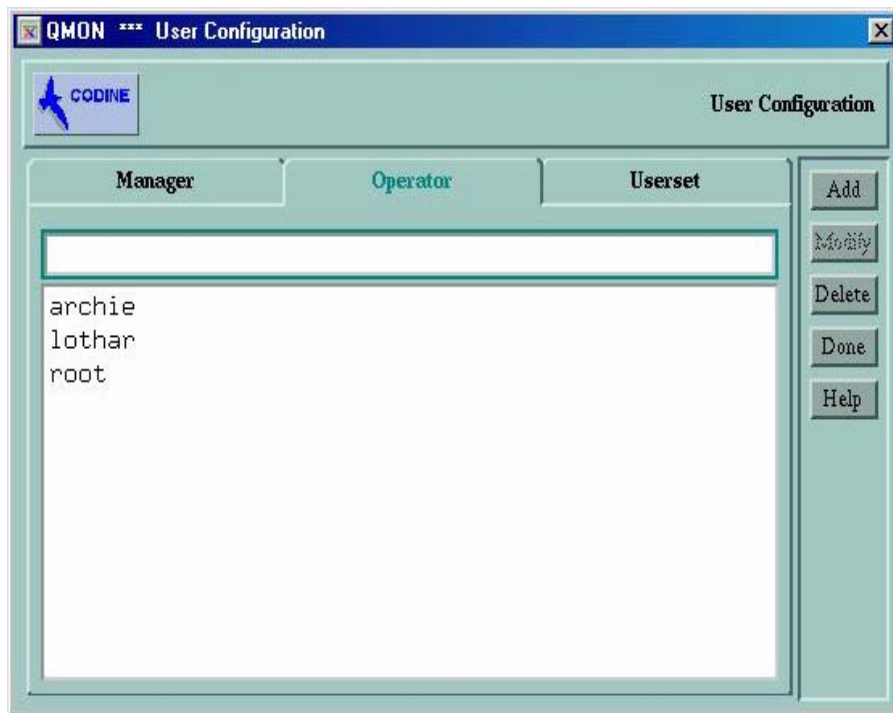


FIGURE 2-45 Operator Configuration dialogue

Configure Operator Accounts from the Command-line

The administration of operator accounts is very similar to those of the Sun Grid Engine managers. The corresponding `qconf` switches are:

```
qconf -ao user_name[,...]
```

add operator. Adds one or multiple users to the list of Sun Grid Engine operators.

```
qconf -do user_name[,...]
```

delete operator. Deletes the specified user(s) from the list of Sun Grid Engine operators.

```
qconf -so
```

show operators. Show the list of all Sun Grid Engine operators.

Queue Owner Accounts

Queue owners are defined during configuration or modifications of a Sun Grid Engine queue. Refer to section "Configuring Queues" on page 75 for a description on how to define queues both with `qmon` and from command-line. Being the owner of a queue is required to be able to

- suspend (stop execution of all jobs running in the queue and close the queue),
- unsuspend (resume execution in the queue and open the queue),
- disable (close the queue, but do not affect running jobs) or
- enable (open the queue)

a queue.

Note – Jobs, which have been suspended explicitly while a queue was suspended will not resume execution when the queue is unsuspended. They need to be unsuspended explicitly.

Typically, users are setup to be owners of certain queues, if these users need certain machines from time to time for important work and if they are affected strongly by Sun Grid Engine jobs running in the background.

User Access Permissions

Any user having a valid login on at least one submit host and an execution host has the possibility to use Sun Grid Engine. However, Sun Grid Engine managers can inhibit access for certain users to certain or all queues. Furthermore, the usage of facilities like specific parallel environments (see section “Support of Parallel Environments” on page 145) can be restricted as well.

For the purpose of defining access permissions, so called *user access lists*, which constitute named arbitrary overlapping or non-overlapping sets of users, have to be defined. User names and UNIX group names can be used to define those user access lists. The user access lists are then used in the cluster configuration (see section “Cluster Configuration” on page 70), in the queue configuration (see section “Configuring Subordinate Queues” on page 84) or in the process of configuring parallel environment interfaces (see section “Configuring PEs with qmon” on page 145) to either deny or allow access to a specific resource.

Configure User Access Lists with qmon

The *Userset* dialogue (see figure 2-46) is opened upon pushing the *User Configuration* button in the *qmon* main menu and selecting the *Userset* tab on the top of the screen. The already available access lists are displayed in the *Usersets* selection list on the left side of the screen. The contents of an access list is displayed in the display region entitled with *Users/Groups* if it is selected by clicking on it with the left mouse button in the *Access Lists* selection list.

Note – Groups are differentiated from users by a prefixed @ sign.

A selected access list can be deleted by pressing the *Delete* button on the right side of the screen. Selected access lists can be modified after pushing the *Modify* button and new access lists can be added after pushing the *Add* button. In both cases, the access list definition dialogue displayed in figure 2-47 is opened and provides the corresponding means:

The *Userset Name* input window either displays the name of the selected access list in the case of a modify operation or can be used to enter the name of the access list to be declared. The *Users/Groups* display region again contains the access list entries as defined so far, while the *User/Group* input window has to be used to add new entries to the access list. The entered user or group names (groups are again prefixed by a @ sign) are appended to the *Users/Groups* display region after pressing the <return> key on the keyboard. Entries can be deleted by selecting them in the display region and pushing the garbage bin icon button.

The modified or newly defined access lists are registered as soon as the Ok button is pressed, or they are discarded if the Cancel button is used instead. In both cases, the access list definition dialogue is closed.

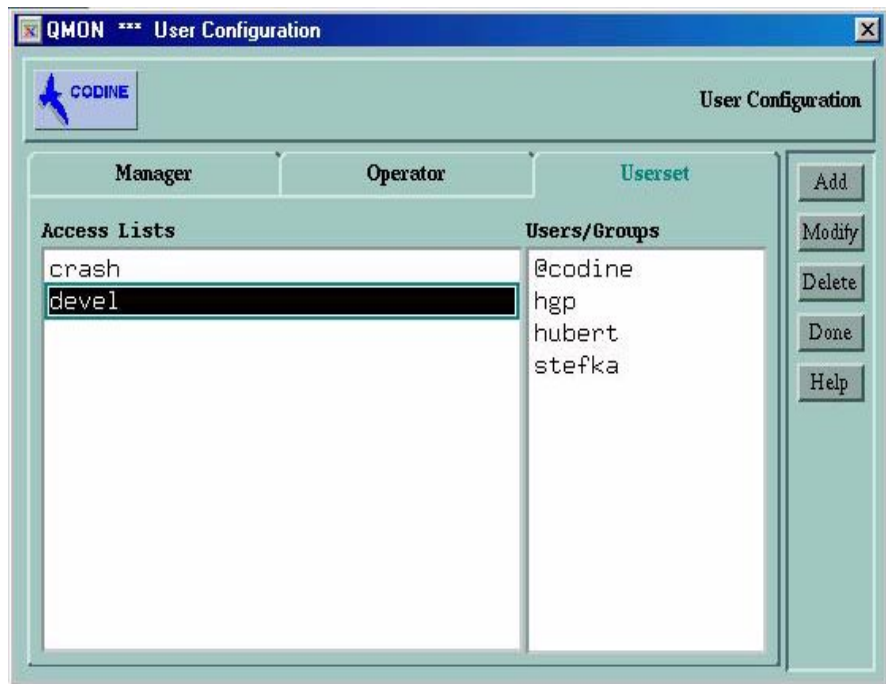


FIGURE 2-46 Userset Configuration

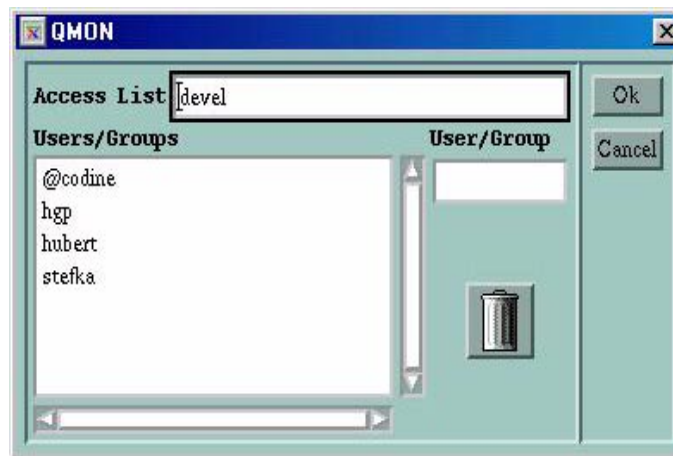


FIGURE 2-47 Access List definition dialogue

Configure User Access from the Command-line

The following options to the `qconf` command can be used to create and maintain user access list from the command-line:

```
qconf -au user_name[,...] access_list_name[,...]
```

add user. Adds one or multiple users to the specified access list(s).

```
qconf -du user_name[,...] access_list_name[,...]
```

delete user. Deletes one or multiple users from the specified access list(s).

```
qconf -su access_list_name[,...]
```

show user access list. Display the specified access list(s).

```
qconf -sul
```

show user access lists. Print a listing of all access lists currently defined.

Scheduling

Overview

Sun Grid Engine's job scheduling activities comprise

- pre-dispatching decisions — such as eliminating execution queues because they are full or overloaded and spooling jobs currently not eligible for execution in a waiting area.
- dispatching — deciding a job's importance with respect to all other pending and running jobs, sensing the load on all the machines in the cluster, and sending the job to an execution queue on a machine selected according to the configured selection criteria,

Sun Grid Engine schedules jobs across a heterogeneous cluster of computers based on

- the cluster's current load,
- the jobs' resource requirements (e.g., CPU, memory, and I/O bandwidth).

Scheduling decisions are based on the strategy for the site and the instantaneous load characteristics of each computer in the cluster. A site's scheduling strategy is expressed through Sun Grid Engine's configuration parameters. Load characteristics are ascertained by collecting performance data as the system runs.

Scheduling Strategies

The administrator can setup strategies with respect to the following Sun Grid Engine scheduling tasks:

- *Queue sorting* - rank the queues in the cluster according to the order in which the queues should be filled up.
- *Job sorting* - determine the order in which Sun Grid Engine attempts to schedule jobs.

Queue sorting

The following means are provided to determine the order in which Sun Grid Engine attempts to fill up queues:

- *Load reporting* - Sun Grid Engine administrators can select, which load parameters are used to compare the load status of hosts and their queues. The wide variety of standard load parameters being available and an interface for extending this set with site-specific load sensors are described under "Load Parameters" on page 113.
- *Load scaling* - Load reports from different hosts can be normalized to reflect a comparable situation (see section "Execution Hosts" on page 62).
- *Load adjustment* - Sun Grid Engine can be configured to automatically correct the last reported load as jobs are dispatched to hosts. The corrected load will represent the expected increase in the load situation caused by recently started jobs. This artificial increase of load can be automatically reduced as the load impact of these jobs shows effect.
- *Sequence number* - Queues can be sorted following a strict sequence.

Job sorting

Before Sun Grid Engine starts dispatching, jobs are brought into an order of highest priority first. Sun Grid Engine will then attempt find suitable resources for the jobs in priority sequence. Without any administrator influence the order is *first-in-first-out* (FIFO). The administrator has the following means of control over the job order:

- *User sort* - If this scheduling alternative is in effect jobs of different users are interleaved. I.e., the first jobs all users have submitted are treated equally, then the second, and so on.

- *Job priority* -Administrators can assign a priority number to the a job thereby directly determining the sorting order. User can lower the priority assigned to their own jobs.
- *Maximum number of user/group jobs* - The maximum number of jobs a user or a Unix user group can have running in the Sun Grid Engine system concurrently can be restricted. This will influence the pending job list sorting order, because jobs of users not exceeding their limit will be given preference.

What Happens in a Scheduler Interval

The Scheduler schedules work in intervals. Between scheduling actions Sun Grid Engine keeps information about significant events such as job submittal, job completion, job cancellation, an update of the cluster configuration, or registration of a new machine in the cluster. When scheduling occurs, the scheduler

- takes into account all significant events,
- sorts jobs and queues corresponding to the administrator specifications,
- takes into account all jobs' resource requirements.

Then, as needed, Sun Grid Engine

- dispatches new jobs,
- suspends executing jobs,
- maintains the status quo.

Scheduler Monitoring

If a job does not get started and if the reasons are unclear to you, you can execute `qalter` for the job together with the `-w v` option. Sun Grid Engine will assume an empty cluster and will check whether there is any queue available which is suitable for the job.

Further information can be obtained by executing `qstat -j job_id`. It will print a summary of the job's request profile containing also the reasons why the job was not scheduled in the last scheduling run. Executing `qstat -j` without a job ID will summarize the reasons for all jobs not having been scheduled in the last scheduling interval.

Note – Collection of scheduling reason information has to be switched on in the scheduler configuration `sched_conf`. Please refer to either the `schedd_job_info` parameter in the corresponding *Sun Grid Engine Reference Manual* manual page or the section "Changing the Scheduler Configuration via `qmon`" on page 131.

To retrieve even further detail about the decisions of the Sun Grid Engine scheduler `cod_schedd`, the option `-tsm` of the `qconf` command can be used. This command will force `cod_schedd` to write trace output to the file

Scheduler Configuration

Default Scheduling

The default Sun Grid Engine scheduling is a *first-in-first-out* policy, i.e. the first job being submitted is the first the scheduler examines in order to dispatch it to a queue. If the first job in the list of pending jobs finds a suitable and idle queue it will be started first in a scheduler run. Only if the first job fails to find a suitable free resource the second job or a job ranked behind may be started before the first in the pending jobs list.

As far as the queue selection for jobs is concerned, the default Sun Grid Engine strategy is to select queues on the least loaded host as long as they deliver suitable service for the job's resource requirements. If multiple suitable queues share the same load the queue being selected is unpredictable.

Scheduling Alternatives

There are various ways to modify the job scheduling and queue selection strategy:

Changing the Scheduling Algorithm

The scheduler configuration parameter `algorithm` (see the `sched_conf` manual page in the *Sun Grid Engine Reference Manual* for further information) is designed to provide a selection for the scheduling algorithm in use. Currently, `default` is the only allowed setting.

Job Priorities

The Sun Grid Engine administration may assign an integer number called job priority to a job spooled in the pending jobs list. The job priority defines the job's position in the pending jobs list. The job with the highest priority number will be examined first by the scheduler. The value range for job priorities is between -1024 and 1023 with 0 being the priority for new jobs just submitted. If a negative priority

value is assigned to a job, the job is sorted even behind new jobs just submitted. If multiple jobs with the same priority number exist the default first-in-first-out rule applies within this priority class.

Job priorities are assigned to a job via the command:

```
% qalter -p prio job_id ...
```

where `prio` specifies the priority to be assigned to the list of jobs as specified in the trailing blank separated Job Id list.

Note – The term job priorities should not be mixed up with the priority queue configuration parameter (see the `queue_conf` manual page in the *Sun Grid Engine Reference Manual*) which defines the `nice` value being set for all jobs executed in a particular queue.

Note – The second column in the `qstat` output shows the priorities currently assigned to the submitted jobs.

Equal Share Sort

The default *first-in-first-out* scheduling policy described above is well known to yield rather unfair results if a user submits a series of jobs one after each other in a short time (e.g. by use of a shell script procedure). The jobs of this user would cover the suitable resources for a very long time offering no chance for other users to allocate these queues.

In this case the cluster administration may change the scheduling policy to the so called *equal share sort*. If this scheduling alternative is in effect and a user already has a running job in the system all his other jobs are sorted behind the jobs of other users in the same priority class (see the previous section for details about priority classes).

The equal share sort is turned on if the scheduler configuration parameter `user_sort` is set to `TRUE` (see the `sched_conf` manual page in the *Sun Grid Engine Reference Manual*).

Scaling System Load

As mentioned above Sun Grid Engine uses the system load information on the machines hosting queues to select the executing queue for a job. This queue selection scheme builds up a load balanced situation thus guaranteeing better utilization of the available resources in a cluster.

However, the system load may not always tell the truth. If, for example, a multi CPU machine is compared to a single CPU system the multiprocessor system usually reports higher load figures as it most probably runs more processes and the system load is a measurement strongly influenced by the number of processes trying to get CPU access. But, multi CPU systems are capable of satisfying a much higher load than single CPU machines. This problem is addressed by processor number adjusted sets of load values which are reported by default by `cod_execd` (see section “Load Parameters” on page 113 and the file `<codine_root>/doc/load_parameters.asc` for details). Use these load parameters instead of the raw load values to avoid the problem described above.

Another example for potentially improper interpretation of load values are systems with strong differences in their performance potential or in their price performance ratio for both of which equal load values do not mean that arbitrary hosts can be selected to execute a job. In this kind of situation, the Sun Grid Engine administrator should define load scaling factors for the concerning execution hosts and load parameters (see section “Execution Hosts” on page 62).

Note – The scaled load parameters are also used to compare them against the load threshold lists `load_thresholds` and `migr_load_thresholds` (see the `queue_conf` manual page in the *Sun Grid Engine Reference Manual* for details).

A further problem associated with load parameters is the need for an application and site dependent interpretation of the values and their relative importance. The CPU load may be dominant for a certain type of application which is common at a particular site, while the memory load is much more important for another site and for the application profile to which the site’s compute cluster is typically dedicated to. To address this problem, Sun Grid Engine allows the administrator to specify a so called *load formula* in the scheduler configuration file `sched_conf` (please refer to the corresponding *Sun Grid Engine Reference Manual* section for more detail). Site specific information on resource utilization and capacity planning can be taken into account by using site defined load parameters (see section “Adding Site Specific Load Parameters” on page 114) and consumable resources (see section “Consumable Resources” on page 96) in the load formula.

Finally, the time dependency of load parameters needs to be taken into account. The load, which is imposed by the Sun Grid Engine jobs running on a system varies in time, and often, e.g. for the CPU load, requires some amount of time to be reported in the appropriate quantity by the operating system. Consequently, if a job was started very recently, the reported load may not provide a sufficient representation

of the load which is already imposed on that host by the job. The reported load will adapt to the real load over time, but the period of time, in which the reported load is too low, may already lead to an oversubscription of that host. Sun Grid Engine allows the administrator to specify *load adjustment* factors which are used in the Sun Grid Engine scheduler to compensate for this problem. Please refer to the *Sun Grid Engine Reference Manual* dealing with the scheduler configuration file `sched_conf` for detailed information on how to set these load adjustment factors.

Select Queue by Sequence Number

Another way to change the default queue selection scheme is to set the global Sun Grid Engine cluster configuration parameter `queue_sort_method` to `seqno` instead of the default `load` (see the `sched_conf` manual page in the *Sun Grid Engine Reference Manual*). In this case, the system load is no longer used to select queues. Instead, the sequence number as assigned to the queues by the queue configuration parameter `seq_no` (see the `queue_conf` manual page in the *Sun Grid Engine Reference Manual*) is used to define a fixed order between the queue in which they are selected (if they are suitable for the considered job and if they are free).

This queue selection policy may be useful if the machines offering batch services at your site are ranked in a monotonous price per job order: e.g., a job running on machine A costs 1 unit of money while it costs 10 units on machine B and 100 units on machine C. Thus the preferred scheduling policy would be to first fill up host A then host B and only if no alternative remains use host C.

Note – It is not defined which queue is selected if the considered queues all share the same sequence number.

Restrict the Number of Jobs per User or Group

The Sun Grid Engine administrator may assign an upper limit to the number of jobs which are allowed to be run by any user or any UNIX group at any point of time. In order to enforce this feature, please set the `maxujobs` and/or `maxgjobs` as described in the `sched_conf` section of the *Sun Grid Engine Reference Manual*.

Changing the Scheduler Configuration via qmon

The Scheduler Configuration dialogue can be opened via clicking on the Scheduler Configuration button in the qmon main menu. The dialogue is separated into the General Parameters section and the Load Adjustment section between which you can switch via the corresponding tab selectors at the top. The following parameters can be defined with the General Parameters dialogue:

- The scheduling algorithm (see section “Changing the Scheduling Algorithm” on page 127).
- The regular time interval between scheduler runs.
- The maximum number of jobs allowed concurrently to run per user and per Unix group (see section “Restrict the Number of Jobs per User or Group” on page 130).
- The queue sorting scheme - either sorting by load or sorting by sequence number (see section “Select Queue by Sequence Number” on page 130).
- Whether or not Equal Share Sort (User Sort flag) is activated (see section “Equal Share Sort” on page 128).
- Whether job scheduling information is accessible through `qstat -j` or not or whether this information should only be collected for a range of job IDs specified in the attached input field. It is recommended to switch on general collection of job scheduling information only temporarily in case of extremely high numbers of pending jobs.
- The load formula to be used to sort hosts and queues.

A sample General Parameters dialogue might look as shown in figure 2-48 on page 132.

The Load Adjustment dialogue allows definition of:

- The load adjustment decay time.
- A table of load adjustment values in the lower half of the dialogue enlisting all load and consumable attributes for which an adjustment value currently is defined. The list can be enhanced by clicking to the Load or Value button at the top. This will open a selection list with all attributes attached to the hosts (i.e. the union of all attributes configured in the `global`, the `host` and the `administrator` defined complexes). The attribute selection dialogue is shown in figure 2-7 on page 66. Selecting one of the attributes and confirming the selection with the `Ok` button will add the attribute to the `Load` column of the `Consumable/Fixed Attributes` table and will put the pointer to the corresponding `Value` field. Modifying an existing value can be achieved by double-clicking with the left mouse button on the `Value` field. Deleting an attribute is performed by first selecting the corresponding table line with the left mouse button. The selected list entry can be deleted either by typing `CTRL-D` or by clicking the right mouse button to open a deletion box and confirming the deletion.

See “Scaling System Load” on page 129 for background information on load adjustment parameters.

A sample Load Adjustment dialog might look as shown in figure 2-49 on page 133.

Please refer to the `sched_conf` manual page in the *Sun Grid Engine Reference Manual* for further detail on the scheduler configuration.

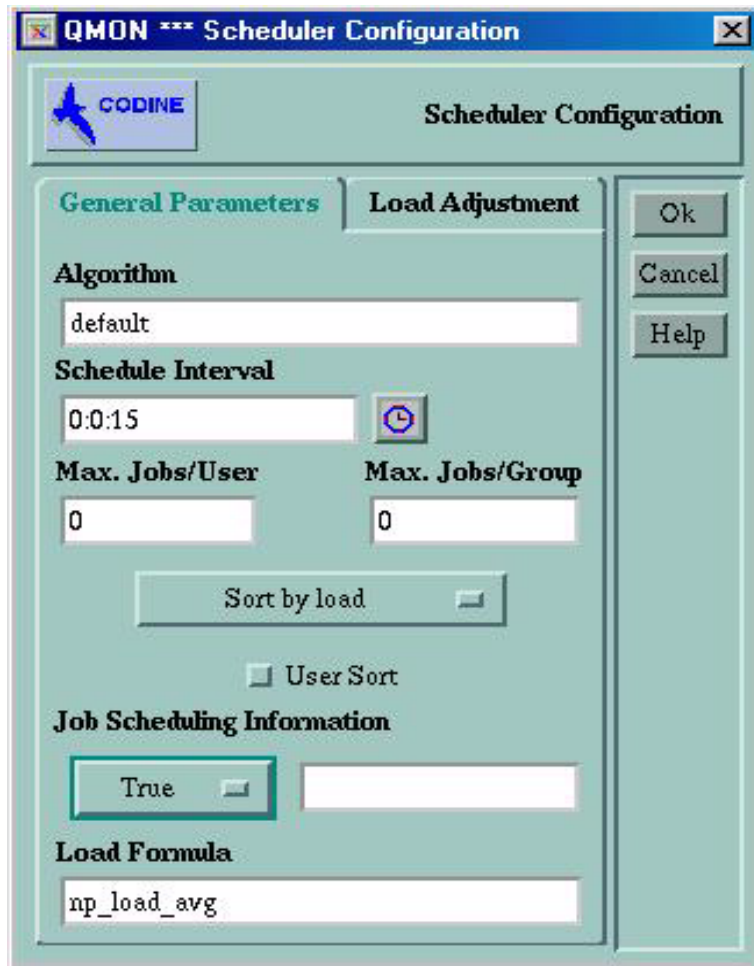


FIGURE 2-48 Scheduler Configuration dialogue “General”

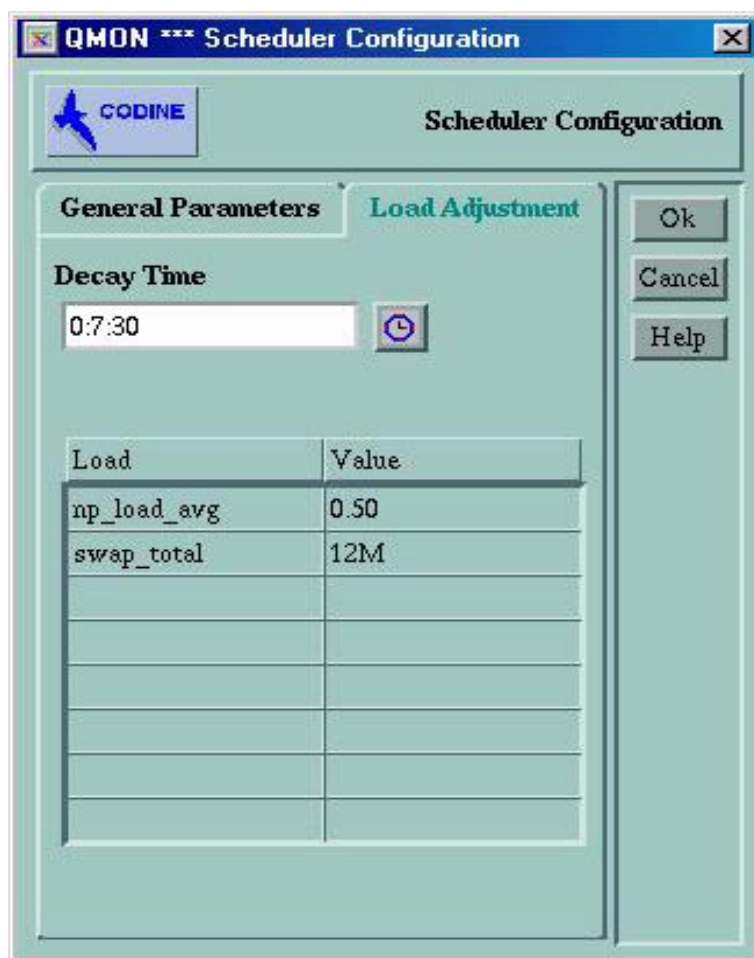


FIGURE 2-49 Scheduler Configuration dialogue "Adjustm."

The Sun Grid Engine Path Aliasing Facility

In networked UNIX environments a user very often has the same home directory (or part of it) on different machines if it has been made accessible across the network via network file system (e.g. NFS). However, sometimes the home directory path is not exactly the same on all machines.

For example, consider user home directories being available via NFS and *automounter*. If a user has a home directory `/home/foo` on the NFS server he will be able to access the home directory under this path on all properly installed NFS clients running *automounter*, but it is important to notice, that `/home/foo` on a client will be just a symbolic link to `/tmp_mnt/home/foo`, where *automounter* physically mounts the directory from the NFSserver.

If, in such a situation, the user would submit a job on a client from somewhere within the home directory tree accompanying it with the `qsub -cwd` flag (execute job in current working directory) Sun Grid Engine might get into trouble to locate the current working directory on the execution host if it is the NFS server. The reason for this is, that `qsub` will grab the current working directory on the submit host and will get `/tmp_mnt/home/foo/...` as this is the physical location on the submit host. This path will be passed over to the execution host and cannot be resolved if the execution host is the NF server with a physical home directory path of `/home/foo`.

Other occasions usually causing similar problems are fixed (non automounted) NFS mounts with different mount point paths on different machines (e.g. mounting home directories under `/usr/people` on one host and `/usr/users` on another) or symbolic links from outside into a network available file system.

In order to resolve such problems, Sun Grid Engine offers both the administrator and the user the possibility to configure a *path aliasing file*. There is a cluster global path aliasing file under `<codine_root>/<cell>/common/codine_aliases` and a user specific under `$HOME/.codine_aliases`. The cluster global file should be modified by the administrator only. Both files share the same format:

- Blank lines and lines with a '#' sign in the first column are skipped.
- Each line other than a blank line or a line lead by '#' has to contain four strings separated by any number of blanks or tabs.
- The first string specifies a source path, the second a submit host, the third an execution host and the fourth the source path replacement.
- Both the submit and the execution host entries may consist of only a '*' sign which matches any host.

The files are interpreted as follows:

- After `qsub` has retrieved the physical current working directory path, the cluster global path aliasing file is read if present. The user path aliases file is read afterwards as if it were appended to the global file.
- Lines not to be skipped are read from the top of the file one by one while the translations specified by those lines are stored if necessary.
- A translation is stored only if the submit host entry matches the host `qsub` is executed on and if the source path forms the initial part either of the current working directory or of the source path replacements already stored.
- As soon as both files are read the stored path aliasing information is passed along with the submitted job.
- On the execution host, the aliasing information will be evaluated. The leading part of the current working directory will be replaced by the source path replacement if the execution host entry of the path alias matches the executing host. Note, that the current working directory string will be changed in this case and that subsequent path aliases must match the replaced working directory path to be applied.

The following is an example how the NFS/*automounter* problem described above can be resolved with an aliases file entry

```
# cluster global path aliases file
# src-path subm-host exec-host dest-path
/tmp_mnt/ * * /
```

Configuring Default Requests

Batch jobs are normally assigned to queues by the Sun Grid Engine system with respect to a request profile defined by the user for a particular job. The user assembles a set of requests which need to be met to successfully run the job and the Sun Grid Engine scheduler only considers queues satisfying the set of requests for this job.

If a user doesn't specify any requests for a job, the scheduler will consider any queue the user has access to without further restrictions. However, Sun Grid Engine allows for configuration of so called *default requests* which may define resource requirements for jobs even though the user did not specify them explicitly.

Default requests can be configured globally for all users of a Sun Grid Engine cluster as well as privately for any user. The default request configuration is represented in default request files. The global request file is located under

`<codine_root>/<cell>/common/cod_request` while the user specific request file is called `.cod_request` and can be located in the user's home directory or in the current working directory in which the `qsub` command is executed.

If these files are present, they are evaluated for every job. The order of evaluation is as follows:

1. First the global default request file.
2. Then the user default request file in the user's home directory.
3. Then the user default request file in the current working directory.

Note – The requests specified in the job script or supplied with the `qsub` command line have higher precedence as the requests in the default request files (see the *Sun Grid Engine User's Guide* for details on how to request resources for jobs explicitly).

Note – Unintended influence of the default request files can be prohibited by use of the `qsub -clear` option, which discards any previous requirement specifications.

The format of both the local and the global default request files is described below:

- The default request files may contain an arbitrary number of lines. Blank lines and lines with a '#' sign in the first column are skipped.
- Each line not to be skipped may contain any `qsub` option as described in the *Sun Grid Engine Reference Manual*. More than one option per line is allowed. The batch script file and argument options to the batch script are not considered as `qsub` options and thus are not allowed in a default request file.
- The `qsub -clear` option discards any previous requirement specifications in the currently evaluated request file or in request files processed before.

As an example, suppose a user's local default request file is configured as follows:

If the user submitted a batch job using the following command:

```
# Local Default Request File
# exec job on a sun4 queue offering 5h cpu
-l arch=solaris64,s_cpu=5:0:0
# exec job in current working dir
-cwd
```

```
% qsub test.sh
```


the effect would be the same as if the user had specified all `qsub` options directly in the command line:

```
% qsub -l arch=solaris64,s_cpu=5:0:0 -cwd test.sh
```

Note – Like batch jobs submitted via `qsub`, interactive jobs submitted via `qsh` will consider default request files also.

Note – Interactive or batch jobs submitted via `qmon` will also take respect to these request files.

Setting Up a Sun Grid Engine User

The following list describes the necessary/available tasks in order to set up a user for Sun Grid Engine:

- **Required Logins:**

In order to submit a job from host *A* for execution on host *B*, the user has to have identical accounts (i.e. identical user names) on the hosts *A* and *B*. No login is required on the machine where `cod_qmaster` runs.

- **Setting Sun Grid Engine Access Permissions:**

Sun Grid Engine offers the ability to restrict user access to the entire cluster, to queues and parallel environments. Please see section “User Access Permissions” on page 122 for a detailed description.

In addition, a Sun Grid Engine user may get the permission to suspend or enable certain queues (see section “Configuring Owners” on page 86 for more information).

- **File Access Restrictions:**

Sun Grid Engine users need to have read access to the directory `<codine_root>/cell/common`.

Before a Sun Grid Engine job is started, the Sun Grid Engine execution daemon (running as `root`) creates a temporary working directory for the job and changes the ownership of the directory to the job owner (the temporary directory is removed as soon as the job finishes). The temporary working directory is created

under the path defined by the queue configuration parameter *tmpdir* (see the *queue_conf* manual page in the *Sun Grid Engine Reference Manual* for more information).

Please make sure, that temporary directories may be created under the *tmpdir* location, set to Sun Grid Engine user ownership and that the users may write to the temporary directories afterwards.

- Site Dependencies:

By definition, batch jobs do not have a terminal connection. Thus, UNIX commands like *stty* in the command interpreters start-up resource file (e.g. *.cshrc* for *csh*) may lead to errors. Please check for occurrence and avoid such commands as described in "Verifying the Installation" on page 52.

As Sun Grid Engine batch jobs usually are executed off-line, there are only two methods to notify a job owner about error events and the like. One way is to log the error messages to file the other is to send electronic mail (e-mail). Under some rare circumstances (e.g. if the error log file can't be opened) e-mail is the only way to directly notify the user (error messages like these are logged to the Sun Grid Engine system logfile anyway, but usually the user would not look into the system logfile). Therefore, it is advantageous if the electronic mail system is properly installed for Sun Grid Engine users.

- Sun Grid Engine Definition Files:

The following definition files may be set up for Sun Grid Engine users: *Qmon* (the resource file for the Sun Grid Engine X-Windows Motif GUI; see section "Customizing *qmon*" on page 138), *.codine_aliases* (current working directory path aliases; see section "The Sun Grid Engine Path Aliasing Facility" on page 134) and *.cod_request* (default request definition file; see section "Configuring Default Requests" on page 135).

Customizing *qmon*

The OSF/1 Motif graphical user's interface of Sun Grid Engine, *qmon*, can be customized by defining or modifying the corresponding X-windows resources. Basically, there are two ways of customizing, either site dependent or user dependent.

There is a template for the resource customizing in the file *<codine_root>/qmon/Qmon*. The site dependent customizing can be achieved by copying this file to the sites default resources file directory (usually something like */usr/lib/X11/app-defaults*) and by modifying it to your site's needs. Another way is to incorporate the contents of the template file adapted to your needs into the sites default X-windows resource file.

As usual for X-windows resources the user can overwrite the site specific customizing. There are three ways to do this:

1. Create a file named `Qmon` in the users home directory containing corresponding resource definitions (e.g. by copying and modifying the site resource file).
2. Incorporate `qmon` related resource definitions into the users private default X-windows resource definition file (usually `.Xdefaults`).
3. Use the `xrdb -merge file_name` command (see the `xrdb` manual page) to merge the `qmon` related resource definitions contained in the file `file_name` into your current settings. You can do this either interactively or in the users default X-windows start-up script (e.g. `.xinitrc`).

Please refer to the template `Qmon` resources file for a description of the resource attributes which can be modified to customize `qmon`.

Gathering Accounting and Utilization Statistics

The Sun Grid Engine command `qacct` can be used to generate alphanumeric accounting statistics. If invoked without switches `qacct` displays the aggregate utilization on all machines of the Sun Grid Engine cluster as generated by all jobs having finished and being contained in the cluster accounting file `<codine_root>/<cell>/common/accounting`. In this case `qacct` just reports three times in seconds:

- **REAL**

The wallclock time. The time between the job starts and the job finishes.

- **USER**

The CPU time spent in the user processes.

- **SYSTEM**

The CPU time spent in system calls.

Several switches are available to report accounting information about all or certain queues, all or certain users, and the like. It is possible in particular, to request information about all jobs having completed and matching a resource requirement specification expressed with the same `-l` syntax as used with the `qsub` command to submit the job. Please refer to the `qacct` manual page in the *Sun Grid Engine Reference Manual* for more information.

A `qacct` option exists to directly access the complete resource usage information stored by Sun Grid Engine including the information as provided by the `getrusage` system call (please refer to the corresponding manual page):

- `-j [job_id|job_name]`

This option reports the resource usage entry for the job(s) with job-id `job_id` or with job name `job_name` respectively. If no argument is given, all jobs contained in the referenced accounting file are displayed. If a job-id is selected and if more than one entry is displayed, either job-id numbers have wrapped around (the range for job-ids is 1 to 999999) or a checkpointing job having migrated is shown.

Checkpointing Support

Checkpointing is a facility to freeze the status of an executing job or application, save this status (the so called checkpoint) to disk and to restart from that checkpoint later on if the job or application has otherwise failed to complete (e.g. due to a system shutdown). If a checkpoint can be moved from one host to another, checkpointing can be used to migrate applications or jobs in a cluster without considerable loss of computational resources. Hence, dynamic load balancing can be provided by the help of a checkpointing facility.

Sun Grid Engine supports two levels of checkpointing:

1. *User level checkpointing*, in which the provision of the checkpoint generation mechanism is entirely in the responsibility of the user or the application. Examples for user level checkpointing are the periodical writing of restart files encoded in the application at prominent algorithmic steps combined with proper processing of these files upon restart of the application or the use of a checkpoint library which needs to be linked with the application and which thereby installs a checkpointing mechanism.

Note – A variety of third party applications provides an integrated checkpoint facility based on writing of restart files.

Note – Checkpoint libraries are available from the public domain (refer to the *Condor* project of the University of Wisconsin for example) or from hardware vendors.

2. *Kernel level transparent checkpointing*, which has to be provided by the operating system (or enhancements to it) and which can be applied to potentially arbitrary jobs. No source code changes or re-linking of your application needs to be provided to use kernel level checkpointing.

Note – Kernel level checkpointing can be applied to complete jobs, i.e. the process hierarchy created by a job, while user level checkpointing is usually restricted to single programs. Thus, the job in which such programs are embedded needs to properly handle the case if the entire job gets restarted.

Note – Kernel level checkpointing as well as checkpointing based on checkpointing libraries can be very resource consuming because the complete virtual address space in use by the job or application at the time of the checkpoint needs to be dumped to disk. As opposed to this, user level checkpointing based on restart files can restrict the data written to the checkpoint on the important *information* only.

Checkpointing Environments

In order to reflect the different types of checkpointing methods outlined above and the potential variety of derivatives of these methods on different operating system architectures, Sun Grid Engine provides a configurable attribute description for each checkpointing method in use called a *checkpointing environment*. Default checkpointing environments are provided with the Sun Grid Engine distribution and can be modified corresponding to the site's needs.

New checkpointing methods can be integrated in principal, but this may become a challenging task and should be performed only by experienced personnel or your Sun Grid Engine support team.

Configuring Checkpointing Environments with qmon

The Checkpointing Configuration dialogue displayed in figure 2-50 on page 142 shows how the already configured checkpointing environments can be viewed (select one of the checkpointing environment names enlisted in the Checkpoint Objects column and the corresponding configuration will be displayed in the Configuration column) added, modified or deleted (use the corresponding buttons). Select the checkpointing environment to be modified or deleted in the Checkpoint Objects column together with the corresponding button. The selected environment will be deleted if the Delete button is pressed or

the Change Checkpoint Object dialogue (see figure 2-51 on page 143) will be opened with the current configuration of the selected checkpointing environment if the Modify button is used. The same dialogue with a template configuration will be opened if the Add button is pressed. Close the Checkpointing Configuration dialogue with the Done button.

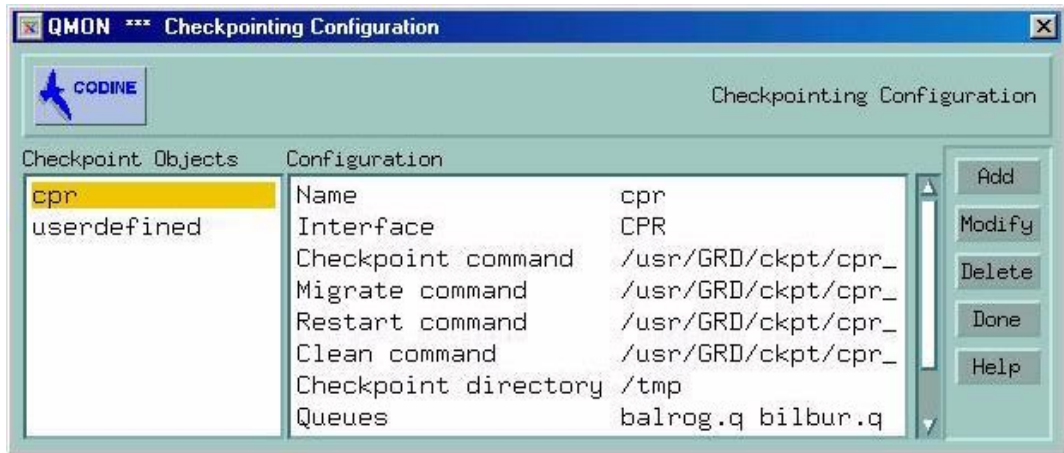


FIGURE 2-50 Checkpointing Configuration dialogue

When pressing the Add or Modify button of the Checkpointing Configuration dialogue (see figure 2-50 on page 142) the Change Checkpoint Object dialogue displayed in figure 2-51 on page 143 is opened. You can define the name of the checkpointing environment to be configured as well as checkpoint/migration/restart/clean-up command strings, a directory where to store checkpoint files to, an occasion specification when checkpoints have to be initiated and a Unix signal to be sent to job/application when a checkpoint is initiated. Please refer to the checkpoint section 5 manual page in the *Sun Grid Engine Reference Manual* for details on these parameters. In addition you have to define the Interface (also called checkpointing method) to be used. Please select one of those provided in the corresponding selection list and refer to the checkpoint manual page for details on the meaning of the different interfaces.

For the checkpointing environments provided with the Sun Grid Engine distribution you should only change the parameters Name, Checkpointing Directory and Queue List. For the latter, please click on the little icon button right to the Queue List window to open the Select Queues dialogue as displayed in figure 2-52 on page 144. Select the queues you want to include in the checkpointing environment from the Available Queues list and add them to the Chosen Queues list. Pressing the Ok button will enter these queues to the Queue List window of the Change Checkpoint Object dialogue.

Note – The queues contained in the queue list of a checkpointing environment need to be of type CHECKPOINTING (see the `queue_conf` manual page for details) to become eligible for the execution of checkpointing jobs.

Use the Ok button in the Change Checkpoint Object dialogue to register your changes with `cod_qmaster` or use the Cancel button to discard your changes.

The screenshot shows a window titled "Change Checkpoint Object". It contains the following fields and controls:

- Name:** A text box containing "cpr".
- Interface:** A button labeled "CPR".
- Queue List:** A list box containing the following items:
 - balrog.q
 - bilbur.q
 - dwain.q
 - fangorn.q
- Checkpoint Command:** A text box containing "/usr/GRD/ckpt/cpr_ckpt_command".
- Migration Command:** A text box containing "/usr/GRD/ckpt/cpr_migration_command".
- Restart Command:** A text box containing "/usr/GRD/ckpt/cpr_restart_command".
- Clean Command:** A text box containing "/usr/GRD/ckpt/cpr_clean_command".
- Checkpointing Directory:** A text box containing "/tmp".
- Checkpoint When:** Three checkboxes:
 - ☒ On Shutdown of Execd
 - ☐ On Min CPU Interval
 - ☐ On Job Suspend
- Checkpoint Signal:** A text box containing "NONE".
- Buttons:** "Ok" and "Cancel" buttons on the right side.

FIGURE 2-51 Change Checkpoint Object

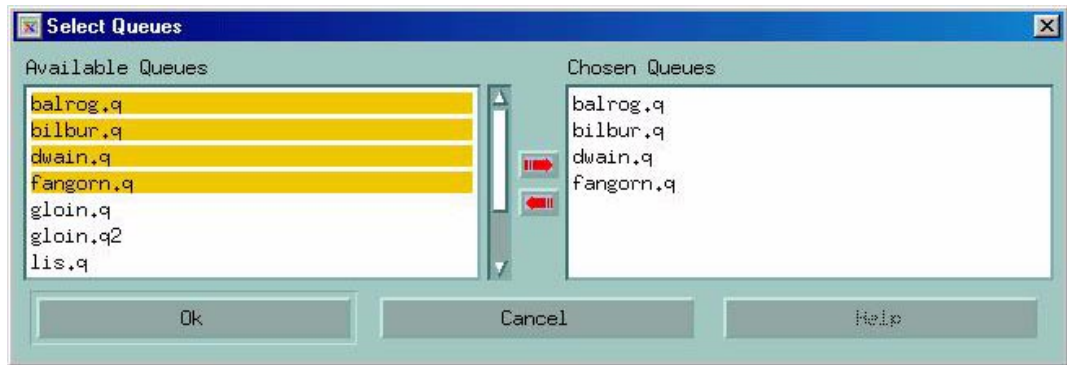


FIGURE 2-52 Checkpointing Queue Selection

Command-line Configuration of Checkpointing Environment.

The following options to the `qconf` command create and maintain checkpointing environment definitions:

`qconf -ackpt ckpt_name`

add checkpointing environment. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with a checkpointing environment configuration template. The parameter `ckpt_name` specifies the name of the checkpointing environment and is already filled into the corresponding field of the template. The checkpointing environment is configured by changing the template and saving to disk. See the checkpoint manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.

`qconf -dckpt ckpt_name`

delete checkpointing environment. Deletes the specified checkpointing environment.

`qconf -mckpt ckpt_name`

modify checkpointing environment. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with the specified checkpointing environment as configuration template. The checkpointing environment is modified by changing the template and saving to disk. See the checkpoint manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.


```
qconf -sckpt ckpt_name
```

show checkpointing environment. Print the configuration of the specified checkpointing environment to standard output.

```
qconf -sckptl
```

show checkpointing environment list. Display a list of the names of all checkpointing environments currently configured.

Support of Parallel Environments

Parallel Environments

A *Parallel Environment* (PE) is a software package designed for concurrent computing in networked environments or parallel platforms. A variety of systems have evolved over the past years into viable technology for distributed and parallel processing on various hardware platforms. Examples for two of the most common message passing environments today are PVM (Parallel Virtual Machine)¹ and MPI (Message Passing Interface)². Public domain as well as hardware vendor provided implementations exist for both tools.

All these systems show different characteristics and have segregative requirements. In order to be able to handle arbitrary parallel jobs running on top of such systems, Sun Grid Engine provides a flexible and powerful interface satisfying the various needs.

Arbitrary PEs can be interfaced by Sun Grid Engine as long as suitable start-up and stop procedures are provided as described in section "The PE Start-up Procedure" on page 150 and in section "Termination of the PE" on page 151, respectively.

Configuring PEs with qmon

The Parallel Environment Configuration dialogue (see figure 2-53) is opened upon clicking with the left mouse button on the PE Config icon button in the qmon main menu. The already configured PEs are displayed in the PE List

1.PVM, Oak Ridge National Laboratories

2.MPI, The Message Passing Interface Forum.

selection list on the left side of the screen. The contents of a PE list is displayed in the display region entitled with Configuration if the PE is selected by clicking on it with the left mouse button in the PE List selection list.

A selected PE list can be deleted by pressing the Delete button on the right side of the screen. Selected PE lists can be modified after pushing the Modify button and new PE lists can be added after pushing the Add button. In both cases, the PE list definition dialogue displayed in figure 2-54 is opened and provides the corresponding means.

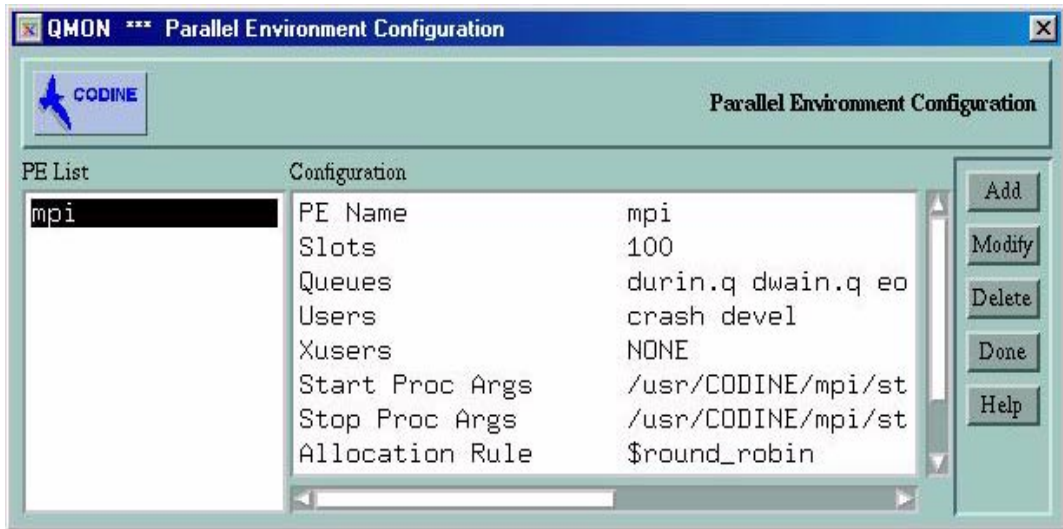


FIGURE 2-53 Parallel Environment Configuration dialogue

The Name input window either displays the name of the selected PE list in the case of a modify operation or can be used to enter the name of the PE list to be declared. The Slots spin box has to be used to enter the number of job slots in total which may be occupied by all PE jobs running concurrently.

The Queue List display region shows the queues which can be used by the PE. By clicking on the little icon button on the right side of the Queue List display region, a Select Queues dialogue as shown in figure 2-55 is opened to modify the PE queue list.

The User Lists display region contains the user access lists (see section “User Access Permissions” on page 122) which are allowed to access the PE while the Xuser Lists display region enlists those access lists, to which access is denied. The little icon buttons associated with both display regions bring up Select Access Lists dialogues as shown in figure 2-55. These dialogues have to be used to modify the content of both access list display regions.

The `Start Proc Args` and `Stop Proc Args` input windows are provided to enter the precise invocation sequence of the PE start-up and stop procedures (see sections "The PE Start-up Procedure" on page 150 and "Termination of the PE" on page 151 respectively). The first argument usually is the start or stop procedure itself. The remaining parameters are command-line arguments to the procedures. A variety of special identifiers (beginning with a '\$' prefix) are available to pass Sun Grid Engine internal run-time information to the procedures. The `sge_pe` manual page in the *Sun Grid Engine Reference Manual* contains a list of all available parameters.

The `Allocation Rule` input window defines the number of parallel processes to be allocated on each machine which is used by a PE. Currently, only positive integer numbers and the special value `$pe_slots` are supported. `$pe_slots` denotes that all processes which are created have to be located on a single host.

The `Control Slaves` toggle button declares whether parallel tasks are generated via Sun Grid Engine (i.e. via `cod_execd` and `cod_shepherd`) or whether the corresponding PE performs its own process creation. It is advantageous if Sun Grid Engine has full control over slave tasks (correct accounting and resource control), but this functionality is only available for PE interfaces especially customized for Sun Grid Engine. Please refer to section "Tight Integration of PEs and Sun Grid Engine" on page 152 for further details.

The `Job is first task` toggle button is only meaningful if `Control Slaves` has been switched on. It indicates, that the job script or one of its child processes acts as one of the parallel tasks of the parallel application (this is usually the case for PVM, for example). If it is switched off, the job script initiates the parallel application but does not participate (e.g. in case of MPI when using "mpirun").

The modified or newly defined PE lists are registered as soon as the `Ok` button is pressed, or they are discarded if the `Cancel` button is used instead. In both cases, the PE list definition dialogue is closed.

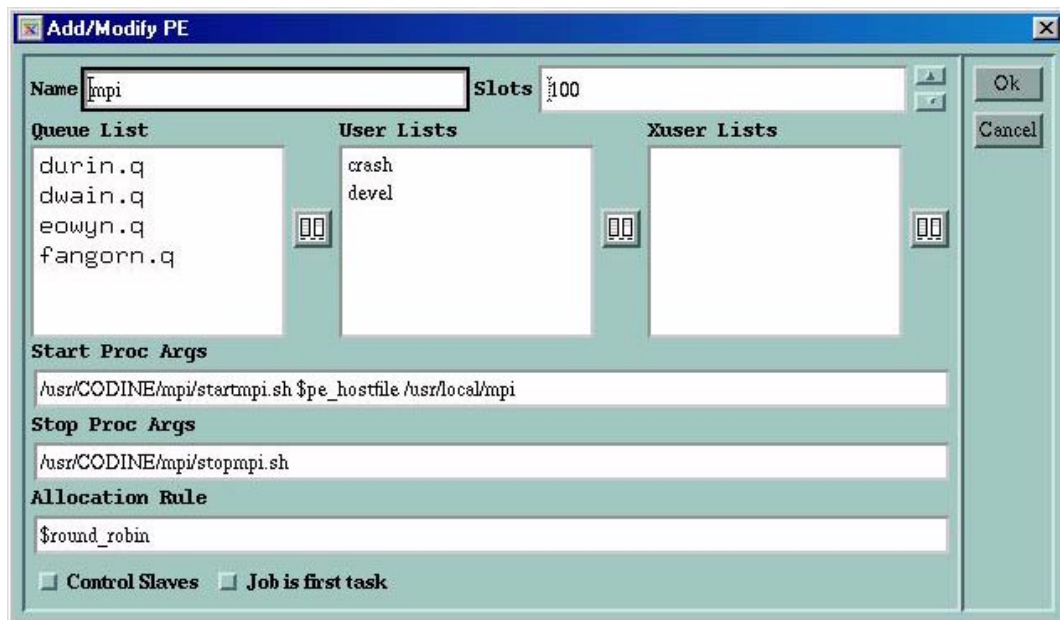


FIGURE 2-54 Parallel environment definition dialogue

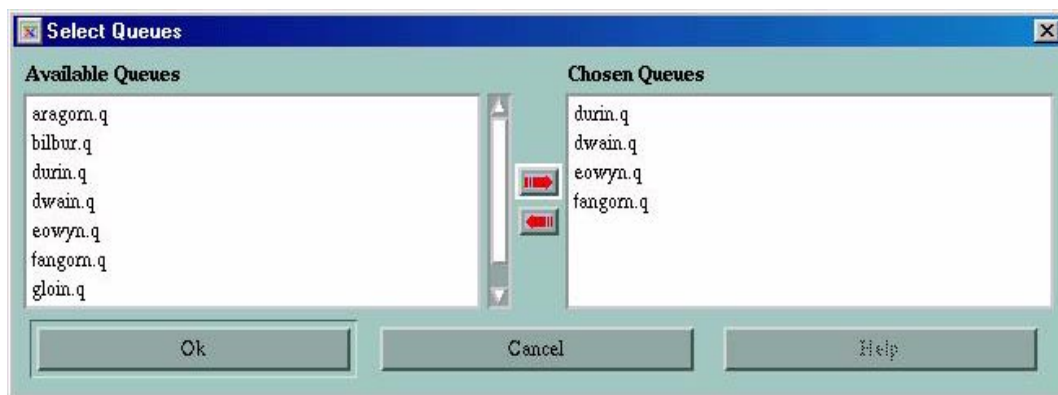


FIGURE 2-55 Select Queues dialogue

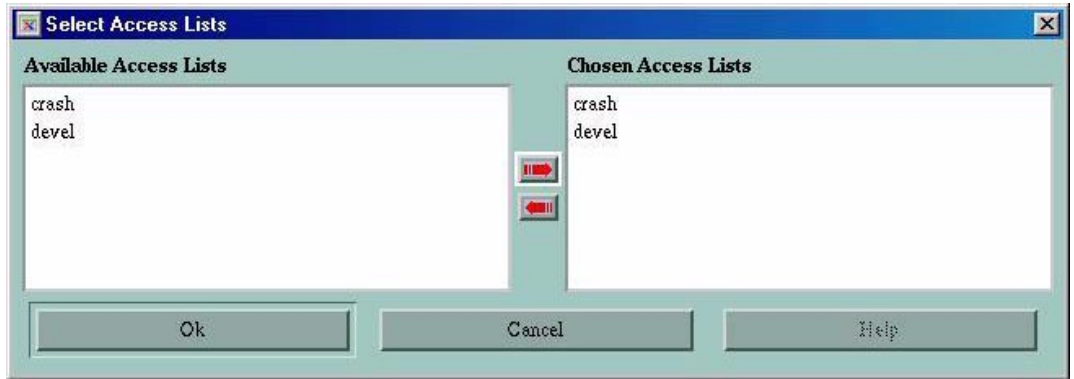


FIGURE 2-56 Select Access Lists dialogue

Configuring PEs from the Command-line

The following options to the `qconf` command create and maintain parallel environment interface definitions:

`qconf -ap pe_name`

add parallel environment. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with a PE configuration template. The parameter *pe_name* specifies the name of the PE and is already filled into the corresponding field of the template. The PE is configured by changing the template and saving to disk. See the `sge_pe` manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.

`qconf -dp pe_name`

delete parallel environment. Deletes the specified PE.

`qconf -mp pe_name`

modify parallel environment. Brings up an editor (default `vi` or corresponding to the `$EDITOR` environment variable) with the specified PE as configuration template. The PE is modified by changing the template and saving to disk. See the `sge_pe` manual page in the *Sun Grid Engine Reference Manual* for a detailed description of the template entries to be changed.

`qconf -sp pe_name`

show parallel environment. Print the configuration of the specified PE to standard output.

```
qconf -spl
```

show parallel environment list. Display a list of the names of all parallel environments currently configured.

The PE Start-up Procedure

Sun Grid Engine starts the PE by simply invoking a start-up procedure via the `exec` system call. The name of the start-up executable and the parameters passed to this executable are configurable from within Sun Grid Engine. An example for such a start-up procedure for the PVM environment is contained in the Sun Grid Engine distribution tree. It consists of a shell script and a C-program which is invoked by the shell script. The shell script uses the C-program to cleanly start-up PVM. All other operations required are handled by the shell script.

The shell script is located under `<codine_root>/pvm/startpvm.sh`. The C-program file can be found under `<codine_root>/pvm/src/start_pvm.c`.

Note – The start-up procedure could have been covered by a single C-program as well. The shell script is used to allow for easier customizing of the sample start-up procedure.

Our example script `startpvm.sh` requires 3 arguments:

1. The path of a host file generated by Sun Grid Engine containing the hostnames, where PVM is going to be started.
2. The host on which the `startpvm.sh` procedure was invoked.
3. The path of the PVM root directory (as usually contained in the `PVM_ROOT` environment variable).

These parameters can be passed to the start-up script via the means described in "Configuring PEs with `qmon`" on page 145. The parameters are among those provided to PE start-up and stop scripts by Sun Grid Engine during runtime. The required host file, as an example, is generated by Sun Grid Engine and the name of the file can be passed to the start-up procedure in the PE configuration by the special parameter name `$codine_hostfile`. A description of all available parameters is given in the `sge_pe` manual page in the *Sun Grid Engine Reference Manual*.

The hostfile has the following format:

- Each line of the file refers to a host on which parallel processes are to be run.
- The first entry of each line denotes the hostname.
- The second entry specifies the number of parallel processes to be run on the host

- The third entry denotes a processor range to be used in case of a multiprocessor machine

This file format is generated by Sun Grid Engine and is fix. PEs, which need a different file format, as for example PVM, need to translate it within the start-up procedure (see `startpvm.sh`).

As soon as the PE start-up procedure has been started by Sun Grid Engine it launches the PE. The start-up procedure should exit with a zero exit status. If the exit status of the start-up procedure is not zero, Sun Grid Engine will report an error and will not start the parallel job.

Note – It is recommended to test any start-up procedures first from the command-line without Sun Grid Engine to remove all errors which may be hard to trace if the procedure is integrated into the Sun Grid Engine framework.

Termination of the PE

When a parallel job finishes or is aborted (via `qdel`) a procedure to halt the parallel environment is called. The definition and semantics of this procedure are very similar to those described for the start-up program. The stop procedure can also be defined in a PE configuration (see for example "Configuring PEs with `qmon`" on page 145).

The stop procedure's purpose is to shutdown the PE and to reap all associated processes.

Note – If the stop procedure fails to clean-up PE processes, Sun Grid Engine may have no information about the processes running under PE control and thus cannot clean-up. Sun Grid Engine, of course, cleans up the processes directly associated with the job script Sun Grid Engine has launched.

The Sun Grid Engine distribution tree also contains an example stop procedure for the PVM PE. It resides under `<codine_root>/pvm/stoppvm.sh`. It takes two arguments:

1. The path to the Sun Grid Engine generated hostfile.
2. The name of the host on which the stop procedure is started.

Likewise the start-up procedure, the stop procedure is expected to return exit status zero on success and a non-zero exit status on failure.

Note – It is recommended to test any stop procedures first from the command-line without Sun Grid Engine to remove all errors which may be hard to trace if the procedure is integrated into the Sun Grid Engine framework.

Tight Integration of PEs and Sun Grid Engine

In section "Configuring PEs with qmon" on page 145 it was mentioned under the explanation of the `Control Slaves` parameter, that PEs for which the creation of parallel tasks is performed by the Sun Grid Engine components `cod_execd` and `cod_shepherd` offer benefits over PEs which perform their own process creation. This is due to the fact that the UNIX operating system allows reliable resource control only for the creator of a process hierarchy. Features like correct accounting, resource limits and process control for parallel applications can only be enforced by the creator of all parallel tasks.

Most PEs do not implement these features and hence do not provide a sufficient interface for the integration with a resource management system like Sun Grid Engine. To overcome this problem Sun Grid Engine provides an advanced PE interface for the tight integration with PEs, which transfers the responsibility for the task creation from the PE to Sun Grid Engine.

The Sun Grid Engine distribution contains two examples of such a tight integration for the PVM public domain version and for the MPICH MPI implementation from Argonne National Laboratories. The examples are contained in the directories `<codine_root>/pvm` and `<codine_root>/mpi` respectively. The directories contain a *loosely* integrated variant of the interfaces for comparison in addition, as well as README files describing the usage and any current restrictions. Please refer to those README files for further detail.

Note – Performing a tight integration with a PE is an advanced task and may require expert knowledge on the PE and the Sun Grid Engine PE interface. You may wish to contact your Sun Grid Engine distributor for support.

The Sun Grid Engine Queuing System Interface (QSI)

Motivation

There are circumstances in which a site does not wish to install Sun Grid Engine on all machines for which batch access has to be provided, but instead wants to use another queuing system already available on these hosts. Typical examples are, that such machines do not belong to the same organization, and thus cannot be maintained by the Sun Grid Engine administration, or that such machines utilize a very special queuing system interfacing specifically designed accounting facilities and the like (very common for so called *Supercomputers*).

In cases like that Sun Grid Engine offers a general interface to such queuing systems. Access to the hosting queuing system (QS) is provided by the concept of *transfer queues*. A transfer queue is defined by the value TRANSFER in the type field of the queue configuration (see section “Configuring Queues” on page 75 in the *Sun Grid Engine Installation and Administration Guide*). The machine hosting a transfer queue is required to provide user-command-style access to the QS. A Sun Grid Engine daemon called `cod_qstd` (Queuing System Transfer Daemon) must run on these gateway machines and a QS interface configuration file, with definitions how to interface the foreign queuing system(s), needs to be provided for `cod_qstd`.

How Jobs for Another Queueing System are Processed

Jobs to be forwarded to another QS can be submitted like any other Sun Grid Engine job. Users request queue attributes for the job via the `qsub` command just like for *normal* Sun Grid Engine jobs (see the *Sun Grid Engine User’s Guide* for details). It is even possible that such a job is processed either within the Sun Grid Engine system or passed outside, depending on the available and best suited resources.

If the Sun Grid Engine scheduler decides to pass the request to another queuing system, i.e. a transfer queue is selected for execution of the job, the necessary information together with the job script is forwarded to the `cod_qstd` on the machine hosting the selected transfer queue. The `cod_qstd` will then generate and execute a submit command to the QS with respect to the definitions in the QSI configuration file (see below) for that host.

Commands like `qstat` or `qdel` will be treated in a similar way. `cod_qstd` provides the necessary mapping between the job as traced by the Sun Grid Engine system (with a unique Sun Grid Engine job-id) and the job as recognized by the other QS. Finishing QS jobs are recognized by `cod_qstd` via a repeatedly executed command, that needs to be provided by the cluster administration and which is defined in the corresponding QSI configuration file. If a job finishes, some accounting information is reported to `cod_qmaster` and a (freely configurable) clean-up procedure is invoked (STDOUT and STDERR output might be transferred by use of the clean-up procedure, for example).

The QSI Configuration File

For each queuing system interfaced by a `cod_qstd` a configuration file needs to be present. These files are expected in `cod_qstd`'s configuration and spool directory (see the `cod_qstd` manual page for details about this spool directory). The names of the configuration files are supposed to start with the string `commands`. (thus `commands_myqs` would be a valid name). At start-up of `cod_qstd` the configuration files need to be present. Each file contains definitions for one QS interfaced by this `cod_qstd`. The format of the files is described in detail in the `qsi_conf` manual page. Here, a brief description of the entries is given:

- `queuing_system`

The name of the QS to be interfaced. The name is arbitrary but needs to correspond with the `queuing_system` queue configuration entry of the transfer queues to pass requests to this QS.

- `transfer_queue`

Attached Sun Grid Engine queue. Sun Grid Engine jobs dispatched to this queue are transferred to this `cod_qstd`. This entry provides the necessary mapping between the queuing system to be interfaced (defined by the parameter above) and the attached transfer queue. See section "Configuring Queues" on page 75 for detailed information on the creation of transfer queues.

- `submit`

The procedure invoked by `cod_qstd` to submit jobs to the QS.

- `delete_job`

If a job passed to the QS is deleted from within Sun Grid Engine via `qdel`, `cod_qstd` executes this procedure.

- `queuing_system_up`

`cod_qstd` regularly executes this procedure to make sure that the QS is up and provides service. If `cod_qstd` notices that the QS is down the status `transfer_down` is reported in the `qstat` output for the transfer queues being configured to forward requests to the QS.

- `job_status`

In order to recognize if jobs running under QS control have finished, `cod_qstd` polls the QS for information using this command. In addition, the special option `-qsi` to the Sun Grid Engine `qstat` command displays status information about forwarded jobs as reported by the QS and this procedure is used to retrieve the information being displayed.

- `job_finished`

This specifies the procedure to be called after QS jobs have finished in order to clean up user data, for example.

- `load_sensor_command`

A user configurable load sensor as described in section "Adding Site Specific Load Parameters" on page 114. The load sensor is intended to measure the load in the foreign queuing system.

- `load_sensor_file`

A file which contains fixed load values. Each line in the file is expected to contain a load parameter name and the blank separated value. If both, a `load_sensor_file` and a `load_sensor_command` are present, the `load_sensor_file` produced values overrule values represented in the `load_sensor_file` in case of equally named load parameters.

The set-up of the procedures to be configured in the QS interface configuration file is crucial for the usability and reliability of the interface. In the following, it is therefore described in more detail.

Setting Up QS Command Procedures

The command procedures used by `cod_qstd` to interface the QS must be set up by the cluster administration. These procedures may be built using arbitrary command interpreters or programming languages, as long as they follow the rules described below:

- A command procedure needs to be a stand-alone executable file provided with the full pathname to the QS interface configuration file.
- The administrator can configure a variety of variables to be expanded at runtime by `cod_qstd` and being passed to a command procedure as command line options. In case of the submit command, for example, available variables are the job script file, the submit directory, the resource limits imposed on the job by the transfer queue configuration and so on (see the `qsi_conf` manual page for details). These parameters need to be processed by the command procedure as defined in the QSI configuration file.

- A command procedure is supposed to show a defined behavior on exit. The submit command, for example, should return with exit status 0 on success and the QS job-id printed on `STDOUT`. On failure it should return with exit status 1 and an error string passed to `STDERR`. Please refer to the `qsi_conf` manual page for details.

Note – `Cod_qstd` checks for the exit status and parses the output of the command procedures. Please make sure that no other output interferes with the one required.

An Example of a QSI file

The following is an example of a QSI configuration file being set up to interface the NQS queuing system and demonstrating a few of the facilities to pass variables to the command procedures:

```
# FILE: commands.nqs - QSI --> NQS
queueing_system nqs
transfer_queue nqs.q
submit /usr/qsi/qsub.sh "$std_err_out" $s_cpu $script_file
delete_job /usr/qsi/qdel.sh $jobid'
suspend_queue not_implemented
queueing_system_up /usr/qsi/qs_up.sh
job_status su /usr/qsi/qstat.sh $jobid'
job_finished /usr/qsi/job_finished.sh $codine_job_id \
              "$std_err_out" "$submitdir" $script_name
load_sensor_command measure_load.sh
load_sensor_file NONE
```

The submit command procedure `qsub.sh` could be provided by the following bourne-shell script:

```
#!/bin/sh
# FILE: qsub.sh; submit command procedure QSI --> NQS
# Processing commandline parameters
ERR_OUT=$1
if [ "$ERR_OUT" = "" ]; then
    # if empty --> default
    ERR_OUT="err_out"
fi
CPU=$2
if [ $CPU = infinity ]; then
    # no nqs-switch in this case
    CPU_SWITCH=""
else
    CPU_SWITCH="-lT $5"
fi
SCRIPT=$3
# Handing off the job to NQS
# The tr-command following the '|' symbol will isolate the
# NQS job ID from qsub's standard output
/usr/nqs/qsub -eo -o $ERR_OUT $CPU_SWITCH \
    $SCRIPT | tr -dc '[0-9\012]'
```

Note – This example only behaves corresponding to the rules, if the exit status of the invoked NQS submit command is 0 on success (1 otherwise) and if the output consists of a single line with no other digits than the NQS job ID (in case of failure an error message should be printed to stderr by the NQS submit). The example submit command procedure needs to be modified correspondingly, if these prerequisites are violated by the NQS derivative being used.

Monitoring QSI Daemons and Jobs

A special switch to the `qstat` command is available to monitor the status of the configured QSIs and of the jobs having been passed to the QS. The switch is `-qsi` and it is available both with or without the `qstat -f` option. The `-qsi` switch takes a

hostname running a `cod_qstd` as optional argument. `Qstat` reports the status of all `cod_qstd`s if a *hostname* is missing and the status of the `cod_qstd` running on the specified host otherwise.

Without the `-f` switch `qstat` displays a brief tabulated listing of the running QS jobs. The `-f` switch forces `qstat` to list a rather complete set of information including the command procedures used to interface the QS.

Trouble Shooting

Scheduler Monitoring

Please refer to section "Scheduler Monitoring" on page 126.

Retrieving Error Reports

Sun Grid Engine reports errors or warnings by logging messages into certain files and/or by electronic mail (e-mail). The logfiles used are:

- Messages Files:

There are separate messages files for the `cod_qmaster`, the `cod_schedd` and the `cod_execd`s. The files have the same file name messages. The `cod_qmaster` logfile resides in the master spool directory, the `cod_schedd` messages file in the scheduler spool directory and the execution daemons' logfiles reside in the spool directories of the execution daemons (see section "Spool Directories Under the Root Directory" on page 44 for more information about the spool directories).

The messages files have the following format:

- Each message occupies a single line.
- The messages are subdivided into 5 components separated by the vertical bar sign (`|`).
- The first component is a time stamp for the message.
- The second specifies the Sun Grid Engine daemon generating the message.
- The third is the hostname the daemon runs on.
- The fourth is a message type which is either N for notice, I for info (both for informational purposes only), W for warning (s.th. may be wrong), E for error (an error condition has been detected) or C for critical (may lead to a program abort).
- The fifth is the message text.

Note – If, for some reason, an error logfile is not accessible, Sun Grid Engine will try to log the error message to the files `/tmp/cod_qmaster_messages`, `/tmp/cod_schedd_messages` or `/tmp/cod_execd_messages` on the corresponding host.

- Job STDERR Output:

As soon as a job is started, the standard error (STDERR) output of the job script is redirected to a file. The file name and the location either complies to a default or may be specified by certain `qsub` command line switches. Please refer to the *Sun Grid Engine User's Guide* and the *Sun Grid Engine Reference Manual* for detailed information.

In some circumstances Sun Grid Engine notifies users and/or administrators about error events via e-mail. The mail messages sent by Sun Grid Engine do not contain a message body. The message text is fully contained in the mail subject field.

Running Sun Grid Engine Programs in Debug Mode

For some severe error conditions the error logging mechanism may not yield sufficient information to identify the problems. Therefore, Sun Grid Engine offers the ability to run almost all ancillary programs and the daemons in *debug* mode. There are different debug levels varying in the extent and depth of information which is provided. The debug levels range from 0 to 10, with 10 being the level delivering the most detailed information and 0 switching off debugging.

To set a debug level an extension to your `.cshrc` or `.profile` resource files is provided with the Sun Grid Engine distribution. For `csh` or `tcsh` users the file `<codine_root>/<util>/dl.csh` is included. For `sh` or `ksh` users the corresponding file is named `<codine_root>/util/dl.sh`. The files need to be “sourced” into your standard resource file. As `csh` or `tcsh` user please include the line:

```
source <codine_root>/util/dl.csh
```

into your `.cshrc` file. As `sh` or `ksh` user, adding the line:

```
. <codine_root>/util/dl.sh
```

into your `.profile` file is the equivalent. As soon as you now logout and login again you can use the following command to set a debug level *level*:

```
% dl level
```

If *level* is greater than 0, starting a Sun Grid Engine command hereafter will force the command to write trace output to `STDOUT`. The trace output may contain warnings, status and error messages as well as the names of the program modules being called internally together with source code line number information (which is helpful for error reporting) depending on the debug level being enforced.

Note – It may be useful to watch a debug trace in a window with a considerable scroll line buffer (e.g. 1000 lines).

Note – If your window is an `xterm` you might want to use the `xterm` logging mechanism to examine the trace output later on.

Running one of the Sun Grid Engine daemons in debug mode will have the result, that the daemons keep their terminal connection to write the trace output. They can be aborted by typing the interrupt character of the terminal emulation you use (e.g. `Control-C`).

Note – To switch off the debug mode, set the debug level back to 0.

User's Guide

Introduction

Sun Grid Engine (Computing in Distributed Networked Environments) is a *load management* tool for heterogeneous, distributed computing environments. Sun Grid Engine provides an effective method for distributing the batch workload among multiple computational servers. In doing so, it increases the productivity of all of the machines and simultaneously increases the number of jobs that can be completed in a given time period. Also, by increasing the productivity of the workstations, the need for outside computational resources is reduced.

Sun Grid Engine provides the user with the means to submit computationally demanding task to the Sun Grid Engine system for transparent distribution of the associated workload. In addition to batch jobs, interactive jobs and parallel jobs can be submitted to Sun Grid Engine. Checkpointing programs are also supported. Checkpointing jobs migrate from workstation to workstation without user intervention on load demand. Comprehensive tools are provided for the monitoring and controlling of Sun Grid Engine jobs.

Please refer to the *Sun Grid Engine Quick Start Guide* for an overview on the Sun Grid Engine system, its features and components. The *Sun Grid Engine Quick Start Guide* also contains a quick installation procedure for a small sample Sun Grid Engine configuration and a glossary of terms commonly used in the Sun Grid Engine manual set.

The *Sun Grid Engine User's Guide* gives an introduction for the user to Sun Grid Engine. The reader is pointed to the *Sun Grid Engine Reference Manual* for a detailed discussion of all available Sun Grid Engine commands. Readers responsible for the cluster administration are pointed to the *Sun Grid Engine Installation and Administration Guide* for a description of the Sun Grid Engine cluster management facilities.

Sun Grid Engine as well as UNIX Commands which can be found in manual pages or the corresponding reference manuals are typeset in emphasized font throughout the *Sun Grid Engine User's Guide*. Command-line in- and output is also typeset in emphasized font and newly introduced or defined terms are typeset in *italics*.

Sun Grid Engine User Types and Operations

There are four user categories in Sun Grid Engine:

1. Managers:

Managers have full capabilities to manipulate Sun Grid Engine. By default, the superusers of any machine hosting a queue have manager privileges.

2. Operators:

The operators can perform the same commands as the manager with the exception of adding/deleting/modifying queues.

3. Owners:

The queue owners are allowed to suspend/enable the owned queues, but have no further management permissions.

4. Users:

Users have certain access permissions as described in "User Access Permissions" on page 171 but no cluster or queue management capabilities. The following table adjoins Sun Grid Engine command capabilities to the different user categories:

TABLE 3-1 Sun Grid Engine Command Capabilities and User Categories

| Command | Manager | Operator | Owner | User |
|---------|---------|--------------------------------------|---|---|
| qacct | Full | Full | Own jobs only | Own jobs only |
| qalter | Full | Full | Own jobs only | Own jobs only |
| qconf | Full | No modifications to the system setup | Show configurations and access permissions only | Show configurations and access permissions only |
| qdel | Full | Full | Own jobs only | Own jobs only |
| qhold | Full | Full | Own jobs only | Own jobs only |
| qhost | Full | Full | Full | Full |
| qlogin | Full | Full | Full | Full |
| qmod | Full | Full | Own jobs and owned queues only | Own jobs only |
| qmon | Full | No modifications to the system setup | No configuration changes | No configuration changes |
| qrexec | Full | Full | Full | Full |
| qselect | Full | Full | Full | Full |
| qsh | Full | Full | Full | Full |
| qstat | Full | Full | Full | Full |
| qsub | Full | Full | Full | Full |

Navigating through the Sun Grid Engine System

Overview on Host Functionality

The Host Configuration button in the qmon main menu allows you to retrieve an overview on the functionality which is associated with the hosts in your Sun Grid Engine cluster. However, unless you do not have Sun Grid Engine manager privileges, you may not apply any changes to the presented configuration.

The host configuration dialogues are described in the *Sun Grid Engine Installation and Administration Guide* in section “Sun Grid Engine Daemons and Hosts” on page 56.

The subsequent sections provide the commands to retrieve this kind of information from the command-line.

The Master Host

The location of the master host should be transparent for the user as the master host may migrate between the current master host and one of shadow master hosts at any time. The file `<codine_root>/<cell>/common/act_qmaster` contains the name of the current master host for the Sun Grid Engine cell `<cell>`.

Execution Hosts

To display information about the hosts being configured as execution hosts in your cluster please use the commands:

```
% qconf -sel
% qconf -se hostname
% qhost
```

The first command displays a list of the names of all hosts being currently configured as execution hosts. The second command displays detailed information about the specified execution host. The third command displays status and load information about the execution hosts. Please refer to the `host_conf` manual page for details on the information displayed via `qconf` and to the `qhost` manual page for details on its output and further options.

Administration Hosts

The list of hosts with administrative permission can be displayed with the command:

```
% qconf -sh
```

Submit Hosts

The list of submit host can be displayed with the command:

```
% qconf -ss
```

Queues and Queue Properties

In order to be able to optimally utilize the Sun Grid Engine system at your site, you should become familiar with the queue structure and the properties of the queues which are configured for your Sun Grid Engine system.

The Queue Control qmon Dialogue

The qmon queue control dialogue displayed and described in section “Controlling Queues with qmon” on page 232 provides a quick overview on the installed queues and their current status.

Show Properties with the qmon Object Browser

The qmon object browser can be used in combination with the queue control dialogue to display the pertinent queue property information. The object browser is opened upon clicking on the **Browser** icon button in the qmon main menu. By selecting the **Queue** button and moving the mouse pointer over a queue icon in the queue control dialogue, queue property information is displayed in a similar way as described in the `queue_conf` manual page

The following figure shows an object browser example display with a queue property print-out.

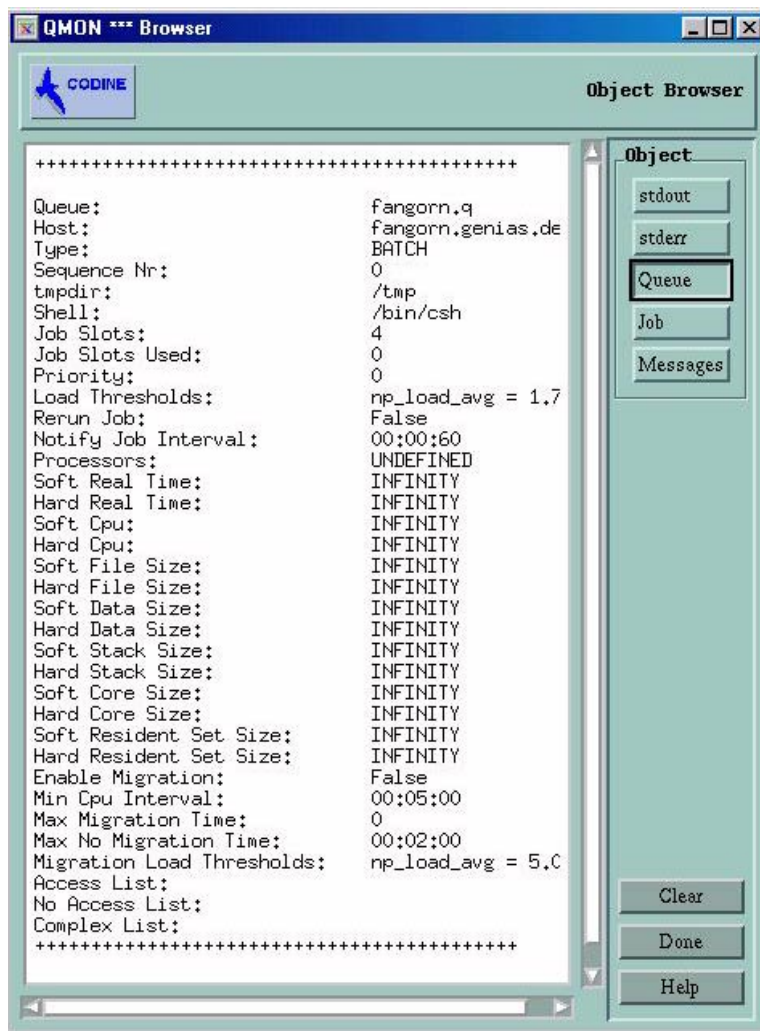


FIGURE 3-1 Browser queue output

Queue Information from the Command-line

In order to display a list of currently configured queues use the

```
% qconf -sql
```

command.

To display the properties of a particular queue please execute:

```
% qconf -sq queue_name
```

A detailed description of each property can be found in the `queue_conf` manual page (see section 5 of the *Sun Grid Engine Reference Manual*). Here is a short introduction to the most important parameters:

- **qname:**
The queue name as requested.
- **hostname:**
The host of the queue.
- **processors:**
The processors of a multi processor system, to which the queue has access.
- **qtype:**
The type of job which is allowed to run in this queue. Currently, this is either batch, interactive, checkpointing, parallel or any combination thereof or transfer alternatively
- **slots:**
The number of jobs which may be executed concurrently in that queue.
- **owner_list:**
The owners of the queue as explained in section “Managers, Operators and Owners” on page 173
- **user_lists:**
The user or group identifiers in the user access lists (see “User Access Permissions” on page 171) enlisted under this parameter may access the queue.
- **xuser_lists**
The user or group identifiers in the user access lists (see “User Access Permissions” on page 171) enlisted under this parameter may *not* access the queue.
- **complex_list**
The complexes enlisted under this parameter are associated with the queue and the attributes contained in these complexes contribute to the set of requestable attributes for the queue (see “Requestable Attributes” on page 168).
- **complex_values**
Assigns capacities as provided for this queue for certain complex attributes (see “Requestable Attributes” on page 168).

Requestable Attributes

When submitting a Sun Grid Engine job a requirement profile of the job can be specified. The user can specify attributes or characteristics of a host or queue which the job requires to run successfully. Sun Grid Engine will map these job requirements onto the host and queue configurations of the Sun Grid Engine cluster and will, therefore, find the suitable hosts for a job.

The attributes which can be used to specify the job requirements are either related to the Sun Grid Engine cluster (e.g. space required on a network shared disk), to the hosts (e.g. operating system architecture), to the queues (e.g. permitted CPU time) or the attributes are derived from site policies such as the availability of installed software only on some hosts.

The available attributes include the queue property list (see “Queues and Queue Properties” on page 165), the list of global and host related attributes (see “Complex Types” on page 90 of the *Sun Grid Engine Installation and Administration Guide*) as well as administrator defined attributes. For convenience, however, the Sun Grid Engine administrator commonly chooses to define only a subset of all available attributes to be requestable.

The attributes being currently requestable are displayed in the Requested Resources sub-dialogue (see figure 3-2 on page 168) to the qmon Submit dialogue (please refer to section “Submit Batch Jobs” on page 173 for detailed information on how to submit jobs). They are enlisted in the Available Resources selection list.



FIGURE 3-2 Requested Resources dialogue

To display the list of requestable attributes from the command-line, you first have to display the list of currently configured *complexes* with the command:

```
% qconf -scl
```

A so called complex contains the definition for a set of attributes. There are three standard complexes: *global* (for the cluster global attributes), *host* (for the host specific attributes and *queue* (for the queue property attributes). Any further complex names printed if the above command is executed refers to an administrator defined complex (see “The Complexes Concept” on page 88 in the *Sun Grid Engine Installation and Administration Guide* or the complex format description in the section 5 of the *Sun Grid Engine Reference Manual* for more information on complexes).

To display the attributes of a particular complex please execute:

```
% qconf -sc complex_name[,...]
```

The output for the queue complex might for example look as shown in table 3-2 on page 169.

TABLE 3-2 “queue” complex

| #name | shortcut | type | value | relop | requestable | consumable | default |
|----------|----------|--------|---------|-------|-------------|------------|---------|
| #----- | | | | | | | |
| qname | q | STRING | NONE | == | YES | NO | NONE |
| hostname | h | HOST | unknown | == | YES | NO | NONE |
| tmpdir | tmp | STRING | NONE | == | NO | NO | NONE |
| calendar | c | STRING | NONE | == | YES | NO | NONE |
| priority | pr | INT | 0 | >= | NO | NO | 0 |
| seq_no | seq | INT | 0 | == | NO | NO | 0 |
| rerun | re | INT | 0 | == | NO | NO | 0 |
| s_rt | s_rt | TIME | 0:0:0 | <= | NO | NO | 0:0:0 |
| h_rt | h_rt | TIME | 0:0:0 | <= | YES | NO | 0:0:0 |
| s_cpu | s_cpu | TIME | 0:0:0 | <= | NO | NO | 0:0:0 |
| h_cpu | h_cpu | TIME | 0:0:0 | <= | YES | NO | 0:0:0 |

TABLE 3-2 “queue” complex

| | | | | | | | |
|------------------|---------|--------|-------|----|-----|----|-------|
| s_data | s_data | MEMORY | 0 | <= | NO | NO | 0 |
| h_data | h_data | MEMORY | 0 | <= | YES | NO | 0 |
| s_stack | s_stack | MEMORY | 0 | <= | NO | NO | 0 |
| h_stack | h_stack | MEMORY | 0 | <= | NO | NO | 0 |
| s_core | s_core | MEMORY | 0 | <= | NO | NO | 0 |
| h_core | h_core | MEMORY | 0 | <= | NO | NO | 0 |
| s_rss | s_rss | MEMORY | 0 | <= | NO | NO | 0 |
| h_rss | h_rss | MEMORY | 0 | <= | YES | NO | 0 |
| min_cpu_interval | mci | TIME | 0:0:0 | <= | NO | NO | 0:0:0 |
| max_migr_time | mmt | TIME | 0:0:0 | <= | NO | NO | 0:0:0 |
| max_no_migr | mnm | TIME | 0:0:0 | <= | NO | NO | 0:0:0 |

#--- # starts a comment but comments are not saved across edits ---

The column name is basically identical to the first column displayed by the `qconf -sq` command. The queue attributes cover most of the Sun Grid Engine queue properties. The `shortcut` column contains administrator definable abbreviations for the full names in the first column. Either the full name or the shortcut can be supplied in the request option of a `qsub` command by the user.

The column `requestable` tells whether the Corresponding entry may be used in `qsub` or not. Thus the administrator can, for example, disallow the cluster’s users to request certain machines/queues for their jobs directly, simply by setting the entries `qname` and/or `qhostname` to be not requestable. Doing this, implies that feasible user requests can be met in general by multiple queues, which enforces the load balancing capabilities of Sun Grid Engine.

The column `relop` defines the relation operation used in order to compute whether a queue meets a user request or not. The comparison executed is:

■ `User_Request relop Queue/Host/...-Property`

If the result of the comparison is false, the user’s job cannot be run in the considered queue. Let, as an example, the queue `q1` be configured with a soft cpu time limit (see the `queue_conf` and the `setrlimit` manual pages for a description of user process limits) of 100 seconds while the queue `q2` is configured to provide 1000 seconds soft cpu time limit.

The columns `consumables` and `default` are meaningful for the administrator to declare so called consumable resources (see section “Consumable Resources” on page 96 of the *Sun Grid Engine Installation and Administration Guide*). The user requests consumables just like any other attribute. The Sun Grid Engine internal bookkeeping for the resources is however different.

Now, let a user submit the following request:

```
% qsub -l s_cpu=0:5:0 nastran.sh
```

The `s_cpu=0:5:0` request (see the `qsub` manual page for details on the syntax) asks for a queue which at least grants for 5 minutes of soft limit cpu time. Therefore, only queues providing at least 5 minutes soft CPU runtime limit are setup properly to run the job.

Note – Sun Grid Engine will only consider workload information in the scheduling process if more than one queue is able to run a job.

User Access Permissions

Access to queues and other Sun Grid Engine facilities (e.g. parallel environment interfaces - see section “Parallel Jobs” on page 196) can be restricted for certain users or user groups by the Sun Grid Engine administrator.

Note – Sun Grid Engine automatically takes into account the access restrictions configured by the cluster administration. The following sections are only important if you want to query your personal access permission.

For the purpose of restricting access permissions, the administrator creates and maintains so called access lists (or in short *ACLs*). The *ACLs* contain arbitrary user and UNIX group names. The *ACLs* are then added to *access-allowed-* or *access-denied-*lists in the queue or in the parallel environment interface configurations (see `queue_conf` or `sge_pe` in *Sun Grid Engine Reference Manual* section 5, respectively).

User’s belonging to *ACLs* which are enlisted in *access-allowed-*lists have permission to access the queue or the parallel environment interface. User’s being members of *ACLs* in *access-denied-*lists may not access the concerning resource.

The Userset Configuration dialogue opened via the User Configuration icon button in the qmon main menu allows you to query for the ACLs you have access to via the Userset Configuration dialogue. Please refer to the section “Managing User Access” on page 117 of the *Sun Grid Engine Installation and Administration Guide* for details.

From the command-line a list of the currently configured ACLs can be obtained by the command:

```
% qconf -sul
```

The entries in one or multiple access lists are printed with the command:

```
% qconf -su acl_name[,...]
```

The ACLs consist of user account names and UNIX group names with the UNIX group names being identified by a prefixed “@” sign. This way you can determine to which ACLs your account belongs.

Note – In case you have permission to switch your primary UNIX group with the `newgrp` command, your access permissions may change.

You can now check for those queues or parallel environment interfaces to which you have access or to which access is denied for you. Please query the queue or parallel environment interface configuration as described in “Queues and Queue Properties” on page 165 and “Configuring PEs with qmon” on page 145 in the *Sun Grid Engine Installation and Administration Guide*. The access-allowed-lists are named `user_lists`. The access-denied-list have the names `xuser_lists`. If your user account or primary UNIX group is associated with a access-allowed-list you are allowed to access the concerning resource. If you are associated with a access-denied-list you may not access the queue or parallel environment interface. If both lists are empty every user with a valid account can access the concerning resource.

Managers, Operators and Owners

A list of Sun Grid Engine managers can be obtained by:

```
% qconf -sm
```

and a list of operators by:

```
% qconf -so
```

Note – The superuser of a Sun Grid Engine administration host is considered as manager by default.

The users, which are owners to a certain queue are contained in the queue configuration database as described in section “Queues and Queue Properties” on page 165. This database can be retrieved by executing:

```
% qconf -sq queue_name
```

The concerning queue configuration entry is called `owners`.

Submit Batch Jobs

Shell Scripts

Shell scripts, also called batch jobs, are in principal a sequence of UNIX command-line instructions assembled in a file. Script files are made executable by the UNIX `chmod` command. If scripts are invoked, a proper command interpreter is started (e.g. `csh`, `tcsh`, `sh`, or `ksh`) and each instruction is interpreted as typed in manually by the user executing the script. Arbitrary UNIX commands, applications and other shell scripts can be invoked from within a shell script.

The appropriate command interpreter is either invoked as `login-shell` or not depending whether its name (`csch`, `tcsh`, `sh`, `ksh`, ...) is contained in the value list of the `login_shells` entry of the Sun Grid Engine configuration in effect for the particular host and queue executing the job.

Note – Note, that the Sun Grid Engine configuration may be different for the various hosts and queues configured in your cluster. You can display the effective configurations via the `-sconf` and `-sq` options of the `qconf` command (refer to the *Sun Grid Engine Reference Manual* for detailed information).

If the command interpreter is invoked as `login-shell`, the environment of your job will be exactly the same as if you just have logged-in and executed the job-script. In case of using `csch` for example, `.login` and `.cschrc` will be executed in addition to the system default start-up resource files (e.g. something like `/etc/login`) while only `.cschrc` will be executed if `csch` is not invoked as `login-shell`. Refer to the manual page of the command interpreter of your choice for a description of the difference between being invoked as `login-shell` or not.

Example Script File

Below is the listing of a simple shell script, which first compiles the application flow from its Fortran77 source and then executes it:

```
#!/bin/csh

# This is a sample script file for compiling and
# running a sample FORTRAN program under Sun Grid Engine.

cd TEST

# Now we need to compile the program 'flow.f' and
# name the executable 'flow'.

f77 flow.f -o flow
```

Your local system user's guide will provide detailed information about building and customizing shell scripts (you might also want to look at the `sh`, `ksh`, `csch` or `tcsh` manual page). In the following, the *Sun Grid Engine User's Guide* will emphasize on specialities which are to be considered in order to prepare batch scripts for Sun Grid Engine.

In general, all shell scripts that you can execute from your command prompt by hand can be submitted to Sun Grid Engine as long as they do not require a terminal connection (except for the standard error and output devices, which are

automatically redirected) and as long as they do not need interactive user intervention. Therefore, the script given above is ready to be submitted to Sun Grid Engine and will perform the desired action.

Submitting Sun Grid Engine Jobs

Submitting jobs with qmon (Simple Example)

The qmon Job Submission dialogue is either invoked from the qmon main menu or from the qmon job control dialogue. Pressing the Submit icon button in the qmon main menu opens the dialogue as well as pushing the Submit button in the Job Control dialogue. The screen for entering General parameters looks as follows (see section “Submitting Jobs with qmon (Advanced Example)” on page 181 for a discussion of the Advanced parameter screen).

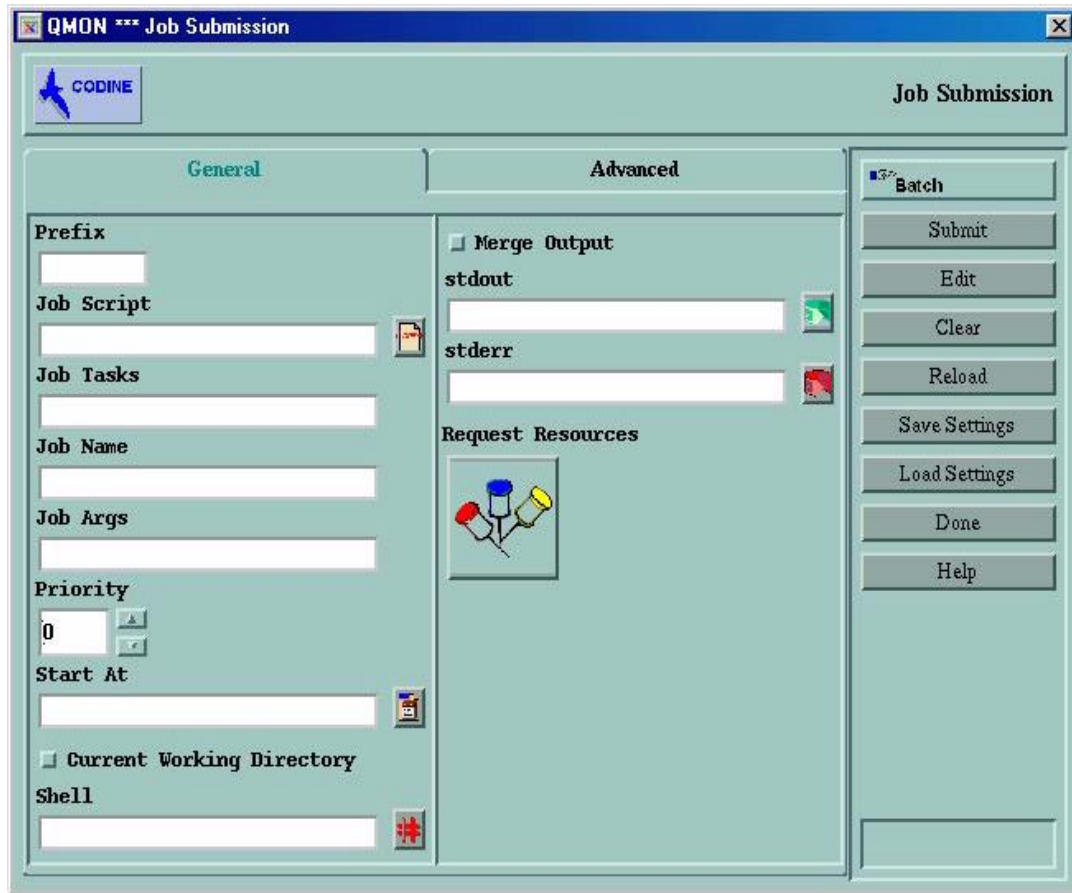


FIGURE 3-3 Job Submission dialogue

Throughout section “Submit Batch Jobs” we will only deal with batch jobs. So please make sure that the default Batch icon is displayed on the top of the button column on the right side of the screen. If an Interactive icon is displayed instead, please click to the icon to change it back to the Batch icon. Please refer to section “Submit Interactive Jobs” on page 200 for detailed information on interactive jobs.

To submit a job you first have to select its script file. Use the file icon button on the right side of the Job Script input window to open the following file selection box and to select the job’s script file.

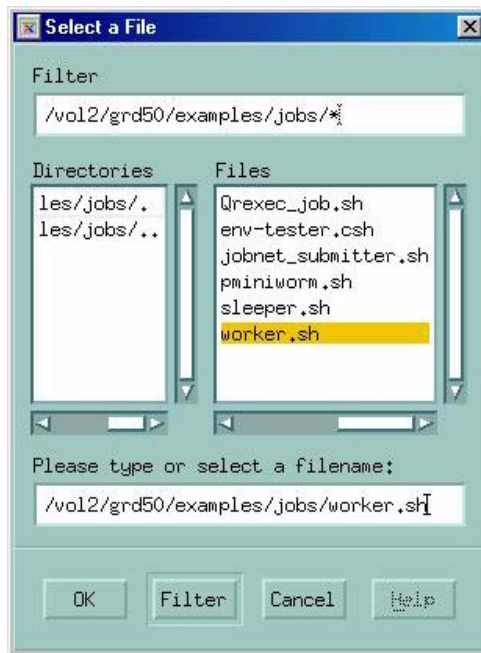


FIGURE 3-4 Job script selection box

Quitting the file selection dialogue with the OK button will transfer the selected file name to the Job Submission dialogue's Job Script input window. Now just click to the Submit button on the right side of the Job Submission screen to submit the job to the Sun Grid Engine system.

Note – To get immediate feedback from the job submission you either need to have the `qmon` Job Control dialogue open (see section “Monitoring and Controlling Jobs with `qmon`” on page 216) or you need the `qmon` Object Browser opened with the display messages facility activated (see section “Additional Information with the `qmon` Object Browser” on page 226).

Submitting jobs with `qmon` (Extended Example)

The standard form of the Job Submission dialogue (see figure 3-15 on page 202) provides the means to configure the following parameters for a job:

- A prefix string which is used for script embedded Sun Grid Engine submit options (please refer to section “Active Sun Grid Engine Comments:” on page 187 for detailed information).

- The job script to be used. If the associated icon button is pushed, a file selection box is opened (see figure 3-4 on page 177)
- The task ID range for submitting array jobs (see “Array Jobs” on page 194).
- The name of the job (a default is set after a job script is selected).
- Arguments to the job script.
- The job’s initial priority value. Users without manager or operator permission may only lower their initial priority value.
- The time at which the job is to be considered eligible for execution. If the associated icon button is pushed, a helper dialogue for entering the correctly formatted time is opened (see figure 3-5 on page 179)
- A flag indicating whether the job is to be executed in the current working directory (for identical directory hierarchies between the submit and the potential execution hosts only).
- The command interpreter to be used to execute the job script (see “How a Command Interpreter Is Selected” on page 185). If the associated icon button is pushed, a helper dialogue for entering the command interpreter specifications of the job is opened (see figure 3-6 on page 179).
- A flag indicating whether the job’s standard output and standard error output are to be merged together into the standard output stream.
- The standard output redirection to be used (see “Output Redirection” on page 186). A default is used if nothing is specified. If the associated icon button is pushed, a helper dialogue for entering the output redirection alternatives (“Output redirection box” on page 179).
- The standard error output redirection to be used. Very similar to the standard output redirection.
- The resource requirements of the job (see “Resource Requirement Definition” on page 191). If resources are requested for a job, the icon button changes its color.
- A selection list button defining whether the job can be restarted after being aborted by a system crash or similar events and whether the restart behavior depends on the queue or is demanded by the job.
- A flag indicating whether the job is to be notified by SIGUSR1 or SIGUSR2 signals respectively if it is about to be suspended or cancelled.
- A flag indicating that either a user hold or a job dependency is to be assigned to the job. The job is not eligible for execution as long as any type of hold is assigned to it (see section “Monitoring and Controlling Sun Grid Engine Jobs” on page 216 for more information concerning holds). The input field attached to the Hold flag allows restricting the hold to only a specific range of task of an array job (see “Array Jobs” on page 194).
- A flag forcing the job to be either started immediately if possible or being rejected. Jobs are not queued, if this flag is selected.

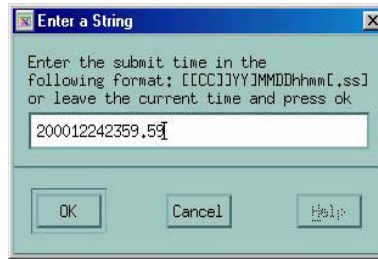


FIGURE 3-5 At time input box

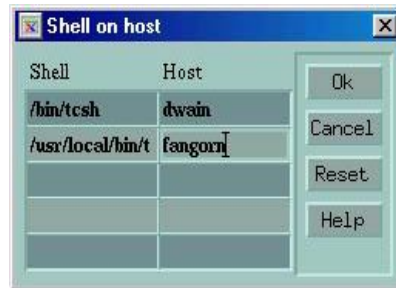


FIGURE 3-6 Shell selection box



FIGURE 3-7 Output redirection box

The buttons at the right side of the Job Submission screen allow you to initiate various actions:

- **Submit**

Submit the job as specified in the dialogue

- **Edit**

Edit the selected script file in an X-terminal either using `vi` or the editor as defined in the `$EDITOR` environment variable.

- Clear

Clear all settings in the Job Submission dialogue including any specified resource requests.

- Reload

Reload the specified script file, parse any script embedded options (see section “Active Sun Grid Engine Comments:” on page 187), parse default settings (see section “Default Requests” on page 190) and discard intermediate manual changes to these settings. This action is the equivalent to a Clear action with subsequent specifications of the previous script file. The option will only show an effect if a script file is already selected.

- Save Settings

Save the current settings to a file. A file selection box is opened to select the file. The saved files may either explicitly be loaded later-on (see below) or may be used as default requests (see section “Default Requests” on page 190).

- Load Settings

Load settings previously saved with the Save Settings button (see above). The loaded settings overwrite the current settings.

- Done

Closes the Job Submission dialogue.

- Help

Dialogue specific help.

Figure “Job submission example” on page 181 shows the submit dialogue with most of the parameters set. The job configured in the example has the script file `flow.sh` which has to reside in the working directory of `qmon`. The job is called `Flow` and the script file takes the single argument `big.data`. The job will be started with priority `-111` and is eligible for execution not before midnight of the 24th of December in the year 2000. The job will be executed in the submission working directory and will use the command interpreter `tcsh`. Finally standard output and standard error output will be merged into the file `flow.out` which will be created in the current working directory also.

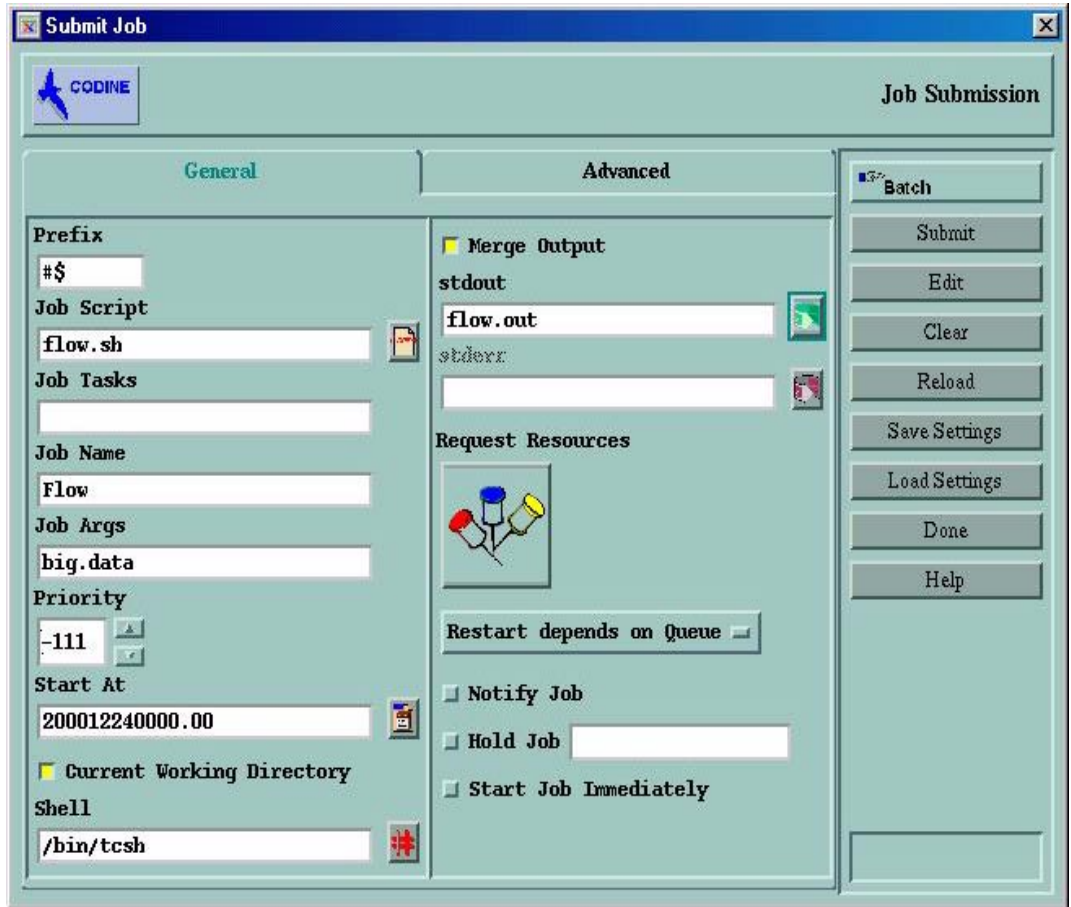


FIGURE 3-8 Job submission example

Submitting Jobs with qmon (Advanced Example)

The Advanced submission screen allows definition of the following additional parameters:

- A parallel environment interface to be used and the range of processes which is required (see section “Parallel Jobs” on page 196).
- A set of environment variables which are to be set for the job before it is executed. If the associated icon button is pushed, a helper dialogue for the definition of the environment variables to be exported is opened (see figure 3-9 on page 183). Environment variables can be taken from qmon’s runtime environment or arbitrary environment variable can be defined.

- A list of name/value pairs called `Context` (see figure 3-10 on page 183), which can be used to store and communicate job related information accessible anywhere from within a Sun Grid Engine cluster. Context variables can be modified from the command-line via the `-ac/-dc/-sc` options to `qsub`, `qsh`, `qlogin` or `qalter` and can be retrieved via `qstat -j`.
- The checkpointing environment to be used in case of a job for which checkpointing is desirable and suitable (see section “Checkpointing Jobs” on page 211).
- An account string to be associated with the job. The account string will be added to the accounting record kept for the job and can be used for later accounting analysis.
- The `Verify` flag, which determines the consistency checking mode for your job. To check for consistency of the job request Sun Grid Engine assumes an empty and unloaded cluster and tries to find at least one queue in which the job could run. Possible checking modes are:
 - `Skip` - no consistency checking at all.
 - `Warning` - inconsistencies are reported, but the job is still accepted (may be desired if the cluster configuration is supposed to change after submission of the job).
 - `Error` - inconsistencies are reported and the job will be rejected if any are encountered.
 - `Just verify` - The job will not be submitted, but an extensive report is generated about the suitability of the job for each host and queue in the cluster.
- The events at which the user is notified via electronic mail. The events `start/end/abortion/suspension` of job are currently defined.
- A list of electronic mail addresses to which these notification mails are sent. If the associated icon button is pushed, a helper dialogue to define the mailing list is opened (see figure 3-11 on page 184).
- A list of queue names which are requested to be the mandatory selection for the execution of the job. The `Hard Queue List` is treated identical to a corresponding resource requirement as described in “Resource Requirement Definition” on page 191.
- A list of queue names which are requested to be a desirable selection for the execution of the job. The `Soft Queue List` is treated identical to a corresponding resource requirement as described in “Resource Requirement Definition” on page 191.
- A list of queue names which are eligible as so called *master queue* for a parallel job. A parallel job is started in the master queue. All other queues to which the job spawns parallel tasks are called *slave queues*.
- An argument list which is forwarded directly to the submission client of a foreign queuing system, in case the job is executed under the Sun Grid Engine QSI (see section “The Sun Grid Engine Queuing System Interface (QSI)” on page 153 in the *Sun Grid Engine Installation and Administration Guide*). The `Transfer QS Arguments` have no effect if the job executed within the Sun Grid Engine system.

- An ID-list of jobs which need to be finished successfully before the job to be submitted can be started. The newly created job *depends* on successful completion of those jobs.

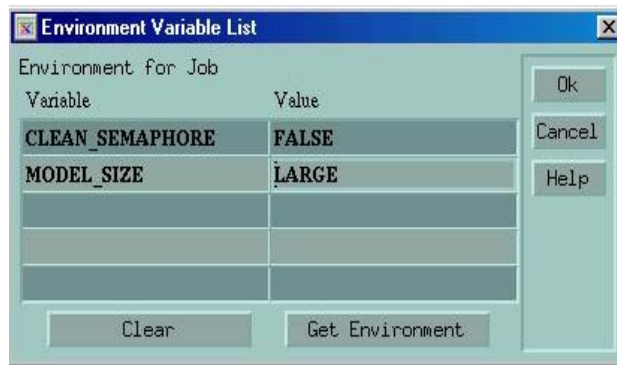


FIGURE 3-9 Job environment definition

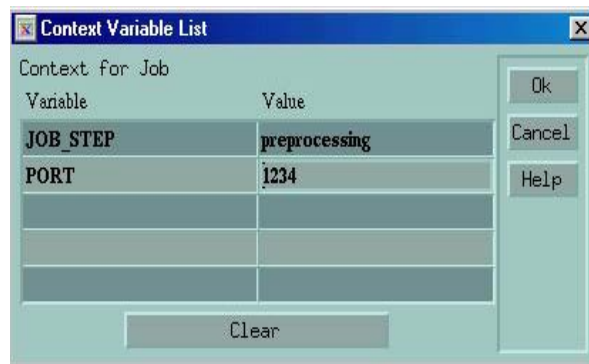


FIGURE 3-10 Job context definition



FIGURE 3-11 Mail address specification

Consequently, the job defined in figure 3-12 on page 185 has the following additional characteristics as compared to the job definition from section "Submitting jobs with `qmon` (Extended Example)" on page 177:

- The job requires the use of the parallel environment `mpi`. It needs at least 4 parallel processes to be created and can utilize up to 16 processes if available.
- Two environment variables are set and exported for the job.
- Two context variables are set.
- The account string `FLOW` is to be added to the job accounting record.
- The job is to be restarted if it fails in case of a system crash.
- Warnings should be printed if inconsistencies between the job request and the cluster configuration are detected
- Mail has to be sent to a list of two e-mail addresses as soon as the job starts and finishes.
- Preferably, the job should be executed in the queue `big_q`.

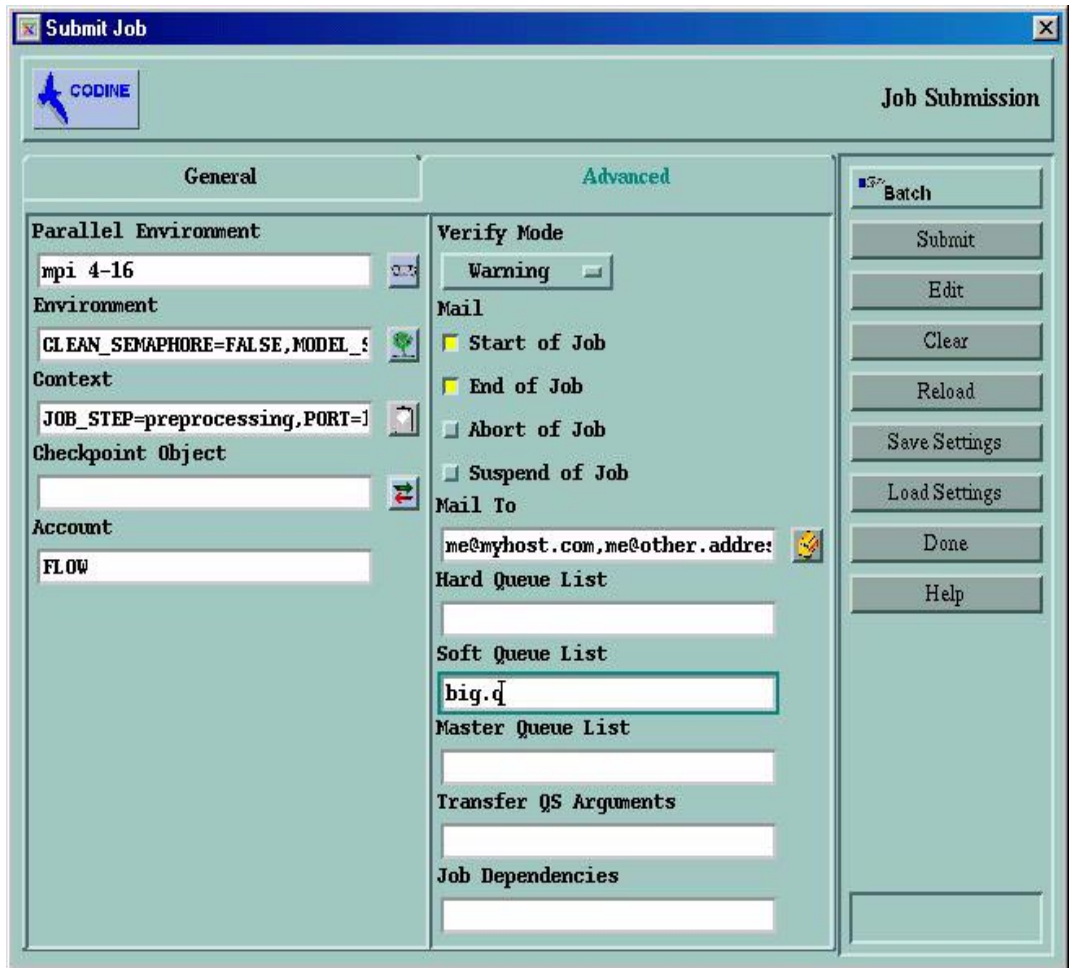


FIGURE 3-12 Advanced job submission example

Extensions to Regular Shell Scripts

There are some extensions to regular shell scripts, that will influence the behavior of the script if running under Sun Grid Engine control. The extensions are:

- How a Command Interpreter Is Selected

The command interpreter to be used to process the job script file can be specified at submit time (see for example page 179). However, if nothing is specified, the configuration variable `shell_start_mode` determines how the command interpreter is selected:

- If `shell_start_mode` is set to `unix_behavior`, the first line of the script file if starting with a „#!“ sequence is evaluated to determine the command interpreter. If the first line has no „#!“ sequence, the Bourne-Shell `sh` is used by default.
- For all other settings of `shell_start_mode` the default command interpreter as configured with the `shell` parameter for the queue in which the job is started is used (see section “Queues and Queue Properties” on page 165 and the `queue_conf` manual page).
- Output Redirection

Since batch jobs do not have a terminal connection their standard output and their standard error output has to be redirected into files. Sun Grid Engine allows the user to define the location of the files to which the output is redirected, but uses defaults if nothing is specified.

The standard location for the files is in the current working directory where the jobs execute. The default standard output file name is `<Job_name>.o<Job_id>`, the default standard error output is redirected to `<Job_name>.e<Job_id>`. `<Job_name>` is either built from the script file name or can be defined by the user (see for example the `-N` option in the `qsub` manual page). `<Job_id>` is a unique identifier assigned to the job by Sun Grid Engine.

In case of array job tasks (see section “Array Jobs” on page 194), the task identifier is added to these filenames separated by a dot sign. Hence the resulting standard redirection paths are `<Job_name>.o<Job_id>.<Task_id>` and `<Job_name>.e<Job_id>.<Task_id>`.

In case the standard locations are not suitable, the user can specify output directions with `qmon` as shown in figure 3-12 and figure 3-7 or with the `-e` and `-o` `qsub` options. Standard output and standard error output can be merged into one file and the redirections can be specified on a per execution host basis. I.e., depending on the host on which the job is executed, the location of the output redirection files becomes different. To build custom but unique redirection file paths, pseudo environment variables are available which can be used together with the `qsub -e` and `-o` option

- `$HOME` - home directory on execution machine.
- `$USER` - user ID of job owner.
- `$JOB_ID` - current job ID.
- `$JOB_NAME` - current job name (see `-N` option).
- `$HOSTNAME` - name of the execution host.
- `$TASK_ID` - array job task index number.

These variables are expanded during runtime of the job into the actual values and the redirection path is built with them.

See the `qsub` manual page in section 1 of the *Sun Grid Engine Reference Manual* for further details.

■ Active Sun Grid Engine Comments:

Lines with a leading “#” sign are treated as comments in shell scripts. Sun Grid Engine, however, recognizes special comment lines and uses them in a special way: the rest of such a script line will be treated as if it were part of the command line argument list of the Sun Grid Engine submit command `qsub`. The `qsub` options supplied within these special comment lines are also interpreted by the `qmon` submit dialogue and the corresponding parameters are preset when a script file is selected.

The special comment lines per default are identified by the “#`$`” prefix string. The prefix string can be redefined with the `qsub -C` option.

The described mechanism is called script embedding of submit arguments. The following example script file makes use of script embedded command-line options.

```
#!/bin/csh
#Force csh if not Sun Grid Engine default shell
#$ -S /bin/csh
# This is a sample script file for compiling and
# running a sample FORTRAN program under Sun Grid Engine.
# We want Sun Grid Engine to send mail when the job begins
# and when it ends.
#$ -M EmailAddress
#$ -m b,e
# We want to name the file for the standard output
# and standard error.
#$ -o flow.out -j y
# Change to the directory where the files are located.
cd TEST
# Now we need to compile the program 'flow.f' and
# name the executable 'flow'.
f77 flow.f -o flow
# Once it is compiled, we can run the program.
flow
```

■ Environment Variables:

When a Sun Grid Engine job is run, a number of variables are preset into the job's environment, as listed below

- ARC: The Sun Grid Engine architecture name of the node on which the job is running. The name is compiled-in into the `cod_execd` binary.
- CODINE_ROOT: The Sun Grid Engine root directory as set for `cod_execd` before start-up or the default `/usr/CODINE`.
- COD_CELL: The Sun Grid Engine cell in which the job executes.
- COD_O_HOME: The home directory path of the job owner on the host from which the job was submitted.
- COD_O_HOST: The host from which the job was submitted.
- COD_O_LOGNAME: The login name of the job owner on the host from which the job was submitted.
- COD_O_MAIL: The content of the MAIL environment variable in the context of the job submission command.
- COD_O_PATH: The content of the PATH environment variable in the context of the job submission command.
- COD_O_SHELL: The content of the SHELL environment variable in the context of the job submission command.
- COD_O_TZ: The content of the TZ environment variable in the context of the job submission command.
- COD_O_WORKDIR: The working directory of the job submission command.
- COD_CHKPT_ENV: Specifies the checkpointing environment (as selected with the `qsub -ckpt` option) under which a checkpointing job executes.
- COD_CHKPT_DIR: Only set for checkpointing jobs. Contains path `ckpt_dir` (see the `checkpoint` manual page) of the checkpoint interface.
- COD_STDERR_PATH: the pathname of the file to which the standard error stream of the job is diverted. Commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start/stop or checkpointing scripts.
- COD_STDOUT_PATH: the pathname of the file to which the standard output stream of the job is diverted. Commonly used for enhancing the output with messages from prolog, epilog, parallel environment start/stop or checkpointing scripts.
- COD_TASK_ID: The task identifier in the array job represented by this task.
- ENVIRONMENT: Always set to BATCH. This variable indicates, that the script is run in batch mode.
- HOME: The user's home directory path from the `passwd` file.
- HOSTNAME: The hostname of the node on which the job is running.
- JOB_ID: A unique identifier assigned by the `cod_qmaster` when the job was submitted. The job ID is a decimal integer in the range to 99999.
- JOB_NAME: The job name, built from the `qsub script filename`, a period, and the digits of the job ID. This default may be overwritten by `qsub -N`.
- LAST_HOST: The name of the preceding host in case of migration of a checkpointing job.

- LOGNAME: The user's login name from the passwd file.
- NHOSTS: The number of hosts in use by a parallel job.
- NQUEUES: The number of queues allocated for the job (always 1 for serial jobs)
- NSLOTS: The number of queue slots in use by a parallel job.
- PATH: A default shell search path of:
/usr/local/bin:/usr/ucb:/bin:/usr/bin
- PE: The parallel environment under which the job executes (for parallel jobs only).
- PE_HOSTFILE: The path of a file containing the definition of the virtual parallel machine assigned to a parallel job by Sun Grid Engine. See the description of the **\$pe_hostfile** parameter in *sge_pe* for details on the format of this file. The environment variable is only available for parallel jobs.
- QUEUE: The name of the queue in which the job is running.
- REQUEST: The request name of the job, which is either the job script filename or is explicitly assigned to the job via the *qsub -N* option.
- RESTARTED: Indicates, whether a checkpointing job has been restarted. If set (to value 1), the job has been interrupted at least once and is thus restarted.
- SHELL: The user's login shell from the passwd file. Note: This is not necessarily the shell in use for the job.
- TMPDIR: The absolute path to the job's temporary working directory.
- TMP: The same as TMPDIR; provided for compatibility with NQS.
- TZ: The time zone variable imported from *cod_execd* if set.
- USER: The user's login name from the passwd file.

Submitting Jobs from the Command-line

Jobs are submitted to Sun Grid Engine from the command-line using the *qsub* command (see the corresponding *Sun Grid Engine Reference Manual* section). A simple job as described in section "Submitting jobs with *qmon* (Simple Example)" on page 175 could be submitted to Sun Grid Engine with the command:

```
% qsub flow.sh
```

if the script file name is *flow.sh*.

As opposed to this, the *submit* command which would yield the equivalent to the *qmon* job submission described in section "Submitting jobs with *qmon* (Extended Example)" on page 177 would look as follows:

```
% qsub -N Flow -p -l11 -a 200012240000.00 -cwd \  
-S /bin/tcsh -o flow.out -j y flow.sh big.data
```

Further command-line options can be added to constitute more complex requests. The job request from section “Submitting Jobs with qmon (Advanced Example)” on page 181, for example, would look as follows:

```
% qsub -N Flow -p -111 -a 200012240000.00 -cwd \  
-S /bin/tcsh -o flow.out -j y -pe mpi 4-16 \  
-v SHARED_MEM=TRUE,MODEL_SIZE=LARGE \  
-ac JOB_STEP=preprocessing,PORT=1234 \  
-A FLOW -w w -r y -m s,e -q big_q\  
-M me@myhost.com,me@other.address \  
flow.sh big.data
```

Default Requests

The last example in the above section demonstrates that advanced job requests may become rather complex and unhandy, in particular if similar requests need to be submitted frequently. To avoid the cumbersome and error prone task of entering such command-lines, the user can either embed qsub options in the script files (see “Active Sun Grid Engine Comments:” on page 187) or can utilize so called *default requests*.

The cluster administration may setup a default request file for all Sun Grid Engine users. The user, on the other hand, can create a private default request file located in the user’s home directory as well as application specific default request files located in the working directories.

Default request files simply contain the qsub options to be applied by default to the Sun Grid Engine jobs in a single or multiple lines. The location of the cluster global default request file is <codine_root>/<cell>/common/cod_request. The private general default request file is located under \$HOME/.cod_request, while the application specific default request files are expected under \$cwd/.cod_request.

If more than one of these files is available, they are merged into one default request with the following order of precedence:

1. Global default request file.
2. General private default request file.
3. Application specific default request file.

Note – Script embedding and the qsub command-line has higher precedence than the default request files. Thus, script embedding overwrites default request file settings, and the qsub command-line options may overwrite these settings again.

Note – The `qsub -clear` option can be used at any time in a default request file, in embedded script commands and in the `qsub` command-line to discard any previous settings.

An example private default request file is presented below:

```
-A myproject -cwd -M me@myhost.com -m b,e  
-r y -j y -S /bin/ksh
```

Unless overwritten, for all jobs of the given user the account string would be *myproject*, the jobs would execute in the current working directory, mail notification would be sent at the beginning and end of the jobs to *me@myhost.com*, the jobs are to be restarted after system crashes, the standard output and standard error output are to be merged and the `ksh` is to be used as command interpreter.

Resource Requirement Definition

In the examples so far the submit options used did not express any requirements for the hosts on which the jobs were to be executed. Sun Grid Engine assumes that such jobs can be run on any host. In practice, however, most jobs require certain prerequisites to be satisfied on the executing host in order to be able to complete successfully. Such prerequisites are enough available memory, required software to be installed or a certain operating system architecture. Also, the cluster administration usually imposes restrictions on the usage of the machines in the cluster. The CPU time allowed to be consumed by the jobs is often restricted, for example.

Sun Grid Engine provides the user with the means to find a suitable host for the user's job without a concise knowledge of the cluster's equipment and its utilization policies. All the user has to do is to specify the requirement of the user's jobs and let Sun Grid Engine manage the task of finding a suitable and lightly loaded host.

Resource requirements are specified via the so called requestable attributes explained in section "Requestable Attributes" on page 168. A very convenient way of specifying the requirements of a job is provided by `qmon`. The Requested Resources dialogue, which is opened upon pushing the Requested Resources icon button in the Job Submission dialogue (see for example figure 3-12 on page 185) only displays those attributes in the Available Resource selection list which currently are eligible. By double-clicking to an attribute, the attribute is added to the Hard or Soft (see below) Resources list of the job and (except for BOOLEAN type attributes, which are just set to True) a helper dialogue is opened to guide the user in entering a value specification for the concerning attribute.

The example Requested Resources dialogue displayed below in figure 3-2 shows a resource profile for a job in which a solaris64 host with an available permas license offering at least 750 Megabytes of memory is requested. If more than one queue fulfilling this specification is found, any defined soft resource requirements are taken into account (none in our example). However, if no queue satisfying both the hard and the soft requirements is found, any queue granting the hard requirements is considered to be suitable.

Note – Only if more than one queue is suitable for a job, load criteria determine where to start the job.



FIGURE 3-13 Requested Resources dialogue

Note – The INTEGER attribute `permas` is introduced via an administrator extension to the “global” complex, the STRING attribute `arch` is imported from the “host” complex while the MEMORY attribute `h_vmem` is imported from the “queue” complex (see section “Requestable Attributes” on page 168)

An equivalent resource requirement profile can as well be submitted from the `qsub` command-line:

```
% qsub -l arch=solaris64,h_vmem=750M,permas=1 \
permas.sh
```

Note – The implicit `-hard` switch before the first `-l` option has been skipped.

The notation `750M` for 750 Megabytes is an example for the Sun Grid Engine quantity syntax. For those attributes requesting a memory consumption you can specify either integer decimal, floating point decimal, integer octal and integer hexadecimal numbers appended by the so called multipliers:

- `k`
multiplies the value by 1000.
- `K`
multiplies the value by 1024.
- `m`
multiplies the value by 1000 times 1000.
- `M`
multiplies the value by 1024 times 1024.

Octal constants are specified by a leading 0 (zero) and digits ranging from 0 to 7 only. Specifying a hexadecimal constant requires to prepend the number by 0x and to use digits ranging from 0 to 9, a to f and A to F. If no multipliers are appended the values are considered to count as bytes. If using floating point decimals, the resulting value will be truncated to an integer value.

For those attributes imposing a time limit one can specify the time values in terms of hours, minutes or seconds and any combination. The hours, minutes and seconds are specified in decimal digits separated by colons. A time of `3:5:11` is translated to 11111 seconds. If a specifier for hours, minutes or seconds is 0 it can be left out if the colon remains. Thus a value of `:5:` is interpreted as 5 minutes. The form used in the Requested Resources dialogue above is an extension, which is only valid within `qmon`.

How Sun Grid Engine Allocates Resources

As shown in the last section, it is important for the user to know, how Sun Grid Engine processes resource requests and how resources are allocated by Sun Grid Engine. The following provides a schematic view of Sun Grid Engine's resource allocation algorithm:

Read in and parse all default request files (see section "Default Requests" on page 190). Process the script file for embedded options (see section "Active Sun Grid Engine Comments:" on page 187). All script embedding options are read, when the job is submitted regardless of their position in the script file. Now read and parse all requests from the command line.

As soon as all qsub requests are collected, *Hard* and *soft* requests are processed separately (the hard first). The requests are evaluated Corresponding to the following order of precedence:

- from left to right of the script/default request file
- from top to bottom of the script/default request file
- from left to right of the command line

In other words, the command line can be used to override the embedded flags.

The resources requested hard are allocated. If a request is not valid, the submit is rejected. If one or more requests cannot be met at submit-time (e.g. a requested queue is busy) the job is spooled and will be re-scheduled at a later time. If all hard requests can be met, they are allocated and the job can be run.

The resources requested soft are checked. The job can run even if some or all of these requests cannot be met. If multiple queues (already meeting the hard requests) provide parts of the soft resources list (overlapping or different parts) Sun Grid Engine will select the queues offering the most soft requests.

The job will be started and will cover the allocated resources.

It is useful to gather some experience on how argument list options and embedded options or hard and soft requests influence each other by experimenting with small test scriptfiles executing UNIX commands like `hostname` or `date`.

Array Jobs

Parametrized and repeated execution of the same set of operations (contained in a job script) is an ideal application for the Sun Grid Engine *array job* facility. Typical examples for such applications are found in the Digital Content Creation industries for tasks like rendering. Computation of an animation is split into frames, in this example, and the same rendering computation can be performed for each frame independently.

The array job facility offers a convenient way to submit, monitor and control such applications. Sun Grid Engine, on the other hand, provides an efficient implementation of array jobs, handling the computations as an array of independent tasks joined into a single job. The tasks of an array job are referenced through an array index number. The indices for all tasks span an index range for the entire array job which is defined during submission of the array job by a single `qsub` command.

An array job can be monitored and controlled (e.g. suspended, resumed or cancelled) as a total or by individual task or subset of tasks, in which case the corresponding index numbers are suffixed to the job ID to reference the tasks. As tasks execute (very much like regular jobs), they can use the environment variable `$COD_TASK_ID` to retrieve their own task index number and to access input data sets designated for this task identifier.

The following is an example of how to submit an array job from the command-line:

```
% qsub -l h_cpu=0:45:0 -t 2-10:2 render.sh data.in
```

The `-t` option defines the task index range. In this case, `2-10:2` specifies that 2 is the lowest and 10 is the highest index number while only every second index (the `:2` part of the specification) is used. Thus the array job consists of 5 tasks with the task indices 2, 4, 6, 8, and 10. Each task requests a hard CPU time limit of 45 minutes (the `-l` option) and will execute the job script `render.sh` once being dispatched and started by Sun Grid Engine. The tasks can use `$COD_TASK_ID` to find out whether they are task 2, 4, 6, 8, or 10 and they can use their index number to find their input data record in the data file `data.in`.

The submission of array jobs from the GUI `qmon` works identically to how it was described in previous chapters. The only difference is, that the Job Tasks input window shown in figure 3-8 on page 181 needs to contain the task range specification with the identical syntax as for the `qsub -t` option. Please refer to the `qsub` manual page in the *Sun Grid Engine Reference Manual* for detailed information on the array index syntax.

The sections “Monitoring and Controlling Sun Grid Engine Jobs” and “Controlling Sun Grid Engine Jobs from the Command-line” as well as the *Sun Grid Engine Reference Manual* sections about `qstat`, `qhold`, `qrls`, `qmod`, and `qdel` contain the pertinent information about monitoring and controlling Sun Grid Engine jobs in general and array jobs in particular.

Note – Array jobs offer full access to all Sun Grid Engine facilities known for regular jobs. In particular they can be parallel jobs at the same time or can have interdependencies with other jobs.

Parallel Jobs

Sun Grid Engine provides means to execute parallel jobs using arbitrary message passing environments such as PVM or MPI (see the *PVM User's Guide* and the *MPI User's Guide* for details) or shared memory parallel programs on multiple slots in single queues or distributed across multiple queues and (for distributed memory parallel jobs) across machines. An arbitrary number of different *parallel environment* (PE) interfaces may be configured concurrently at the same time.

The currently configured PE interfaces can be displayed with the commands:

```
% qconf -spl
% qconf -sp pe_name
```

The first command prints a list of the names of the currently available PE interfaces. The second command displays the configuration of a particular PE interface. Please refer to the *sgc_pe* manual page for details on the PE configuration.

Alternatively, the PE configurations can be queried with the *qmon Parallel Environment Configuration* dialogue (see section “Configuring PEs with *qmon*” on page 145 in the *Sun Grid Engine Installation and Administration Guide*). The dialogue is opened upon pushing the *PE Config* icon button in the *qmon* main menu.

The example from section “Submitting Jobs with *qmon* (Advanced Example)” on page 181 already defines a parallel job requesting the PE interface *mpi* (for *message passing interface*) to be used with at least 4 but up to (and preferably) 16 processes. The icon button to the right of the parallel environment specification window can be used to pop-up a dialogue box to select the desired parallel environment from a list of available PEs (see figure 3-14). The requested range for the number of parallel tasks initiated by the job can be added after the PE name in the PE specification window of the advanced submission screen.

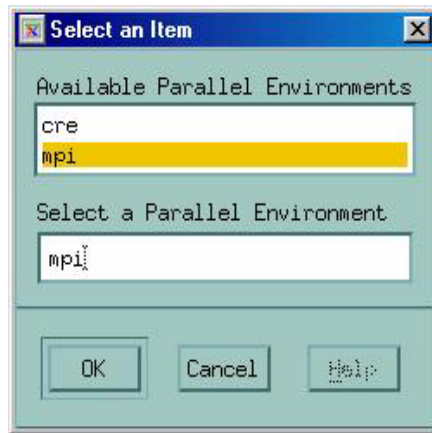


FIGURE 3-14 PE selection

The command-line submit command corresponding to the parallel job specification described above is given in section “Submitting Jobs from the Command-line” on page 189 and shows how the `qsub -pe` option has to be used to formulate an equivalent request. The `qsub` manual page in the *Sun Grid Engine Reference Manual* provides more detail on the `-pe` syntax.

It is important to select a suitable PE interface for a parallel job. PE interfaces may utilize no or different message passing systems, they may allocate processes on single or multiple hosts, access to the PE may be denied to certain users, only a specific set of queues may be used by a PE interface and only a certain number of queue slots may be occupied by a PE interface at any point of time. You should therefore ask the Sun Grid Engine administration for the available PE interface(s) best suited for your type(s) of parallel jobs.

You can specify resource requirements as explained in section “Resource Requirement Definition” on page 191 together with your PE request. This will further reduce the set of eligible queues for the PE interface to those queues also fitting the resource requirement definition you specified. If, for example, the command:

```
% qsub -pe mpi 1,2,4,8 -l nastran,arch=ssf nastran.par
```

is submitted, the queues suitable for this job are those which are associated to the PE interface `mpi` by the PE configuration and also satisfy the resource requirement specification specified by the `qsub -l` option.

Note – The Sun Grid Engine PE interface facility is highly configurable. In particular, the Sun Grid Engine administration can configure the PE start-up and stop procedures (see the `sge_pe` manual page) to support site specific needs. The `qsub -v` and `-V` options to export environment variables may be used to pass information from the user who submits the job to the PE start-up and stop procedures. Please ask the Sun Grid Engine administration if you are required to export certain environment variables.

Submitting Jobs to Other Queueing Systems

Some sites do not wish to install Sun Grid Engine on all machines for which batch access is provided, but instead use other queueing systems already available on these hosts. Typical examples are machines which do not belong to the same organization, and thus cannot be maintained by the Sun Grid Engine administration, or machines utilizing a very special queueing system, interfacing specifically designed accounting facilities and the like (very common for so called *Supercomputers*).

In these cases, Sun Grid Engine offers a general interface to such queueing systems. Access to the hosting queueing system (QS) is provided by the concept of *transfer queues*. A transfer queue is defined by the value `TRANSFER` in the `type` field of the queue configuration (see section “Queues and Queue Properties” on page 165).

Jobs to be forwarded to another QS can be submitted like any other Sun Grid Engine job. Resource requirements are requested for the job via `qmon` or the `qsub` command just like for *normal* Sun Grid Engine jobs. It is even possible that such a job is processed either within the Sun Grid Engine system or passed to a QS, depending on the available and best suited resources.

Sometimes it is necessary to supply QS special switches with the job. To perform this, there are two methods available in the Sun Grid Engine QS interface:

1. Add the options to the script file by usage of special comments similar to the “`#$`” comments in Sun Grid Engine (of course the QS must support such special comments).
2. The special `qsub` option `-qs_args` may be used to pass such options. Everything behind the `-qs_args` option is considered as option to the QS until the `-qs_end` option is encountered. A corresponding input field for such arguments is provided in the `qmon` submission dialogue as well (see section “Submitting Jobs with `qmon` (Advanced Example)” on page 181).

How Sun Grid Engine Jobs Are Scheduled

Job Scheduling

Job Priorities

Concerning the order of scheduling precedence of different jobs a first-in-first-out (fifo) rule is applied by default. I.e., all *pending* (not yet scheduled) jobs are inserted in a list, with the first submitted job being the head of the list, followed by the second submitted job, and so on. The job submitted first will be attempted to be scheduled first. If at least one suitable queue is available, the job will be scheduled. Sun Grid Engine will try to schedule the second job afterwards no matter whether the first has been dispatched or not.

This order of precedence among the pending jobs may be overruled by the cluster administration via a *priority value* being assigned to the jobs. The actual priority value can be displayed by using the `qstat` command (the priority value is contained in the last column of the pending jobs display entitled `P`; refer to section “Monitoring with `qstat`” on page 227 for details). The default priority value assigned to the jobs at submit time is 0. The priority values are positive and negative integers and the pending jobs list is sorted Correspondingly in the order of descending priority values. I.e., by assigning a relatively high priority value to a job, the job is moved to the top of the pending jobs list. Jobs with negative priority values are inserted even after jobs just submitted. If there are several jobs with the same priority value, the fifo rule is applied within that priority value category.

Equal-Share-Scheduling

The fifo rule sometimes leads to problems, especially if user’s tend to submit a series of jobs almost at the same time (e.g. via shell-script issuing one submit after the other). All jobs being submitted afterwards and being designated to the same group of queues will have to wait a very long time. *Equal-share-scheduling* avoids this problem by sorting jobs of users already owning a running job to the end of the precedence list. The sorting is performed only among jobs within the same priority value category. Equal-share-scheduling is activated if the Sun Grid Engine scheduler configuration entry `user_sort` (refer to the `sched_conf` manual page for details) is set to `TRUE`.

Queue Selection

If submitted jobs cannot be run, because requested resources like a queue of a certain group are not available at submit-time, it would be disadvantageous to immediately dispatch such jobs to a certain queue Corresponding to the load average situation. Imagine, a suitable queue is busy with a job, that is terribly slowed down by an infrequently responding I/O device. The machine, hosting this queue, might offer the lowest load average in the Sun Grid Engine cluster, however, the currently executing job might also continue to run for a very long time.

Therefore, Sun Grid Engine does not dispatch jobs requesting **generic** queues if they cannot be started immediately. Such jobs will be marked as spooled at the `cod_qmaster`, which will try to re-schedule them from time to time. Thus, such jobs are dispatched to the next suitable queue, that becomes available.

As opposed to this, jobs which are requested by name to a certain queue, will go directly to this queue regardless whether they can be started or they have to be spooled. Therefore, viewing Sun Grid Engine queues as computer science *batch queues* is only valid for jobs requested by name. Jobs submitted with **generic** requests use the spooling mechanism of `cod_qmaster` for queueing, thus utilizing a more abstract and flexible queuing concept.

If a job is scheduled and multiple free queues meet its resource requests, the job is usually dispatched to the queue (among the suitable) belonging to the least loaded host. By setting the Sun Grid Engine scheduler configuration entry `queue_sort_method` to `seqno`, the cluster administration may change this load dependent scheme into a fixed order algorithm: the queue configuration entry `seq_no` is used to define a precedence among the queues assigning the highest priority to the queue with the lowest sequence number.

Submit Interactive Jobs

Submitting interactive jobs instead of batch jobs is useful in situations where your job requires your direct input to influence the results of the job. This is typically the case for X-windows applications, which are interactive by definition, or for tasks in which your interpretation of immediate results is required to steer the further computation.

Three methods exist in Sun Grid Engine to create interactive jobs:

1. `qlogin` - a telnet like session is started on a host selected by Sun Grid Engine.
2. `qssh` - the equivalent of the standard Unix `rsh` facility. Either a command is executed remotely on a host selected by Sun Grid Engine or a `rlogin` session is started on a remote host if no command was specified for execution.

3. `qsh/qmon` - an `xterm` is brought up from the machine executing the job with the display set corresponding to your specification or the setting of the `DISPLAY` environment variable. If the `DISPLAY` variable is not set and if no display destination was defined specifically, Sun Grid Engine directs the `xterm` to the 0.0 screen of the X server on the host from which the interactive job was submitted.

Note – To function correctly, all the facilities need proper configuration of Sun Grid Engine cluster parameters. The correct `xterm` execution paths have to be defined for `qsh` and interactive queues have to be available for this type of jobs. Please contact your system administrator whether your cluster is prepared for interactive job execution.

The default handling of interactive jobs differs from the handling of batch jobs in that interactive jobs are not queued if they cannot be executed by the time of submission. This is to indicate immediately, that not enough appropriate resources are available to dispatch an interactive job right after it was submitted. The user is notified in such cases that the Sun Grid Engine cluster is too busy currently.

This default behavior can be changed with the `-now no` option to `qsh`, `qlogin` and `qrsh`. If this option is given, interactive jobs are queued like batch jobs. Using `-now yes`, batch jobs submitted with `qsub` also can be handled like interactive jobs and are either dispatched for execution immediately or are rejected.

Note – Interactive jobs can only be executed in queues of the type `INTERACTIVE` (please refer to “Configuring Queues” on page 75 in the *Sun Grid Engine Installation and Administration Guide* for details).

The subsequent sections outline the usage of the `qlogin` and `qsh` facilities. The `qrsh` command is explained in a broader context in chapter “Transparent Remote Execution” on page 204.

Submit Interactive Jobs with `qmon`

The only type of interactive jobs which can be submitted from `qmon` are those bringing up an `xterm` on a host selected by Sun Grid Engine.

By clicking to the icon on top of the button column at the right side of the Job Submission dialogue until the Interactive icon gets displayed, the job submission dialogue is prepared for submitting interactive jobs (see figure 3-15 on page 202 and figure 3-16 on page 203). The meaning and the usage of the selection options in the dialogue is the same as explained for batch jobs in section “Submitting Sun Grid Engine Jobs” on page 175. The basic difference is that several input fields are set insensitive because they do not apply for interactive jobs.

Submit Job

Job Submission

General

Advanced

Prefix

Job Script

Job Tasks

Job Name

INTERACTIVE

Job Args

Priority

0

Start At:

☐ Current Working Directory

Shell

☐ Merge Output

stdout

stderr

Request Resources

Restart depends on Queue

☐ Notify Job

☐ Hold Job

☒ Start Job Immediately

INTERACTIVE

Submit

Edit

Clear

Reload

Save Settings

Load Settings

Done

Help

FIGURE 3-15 Interactive Job Submission dialogue General

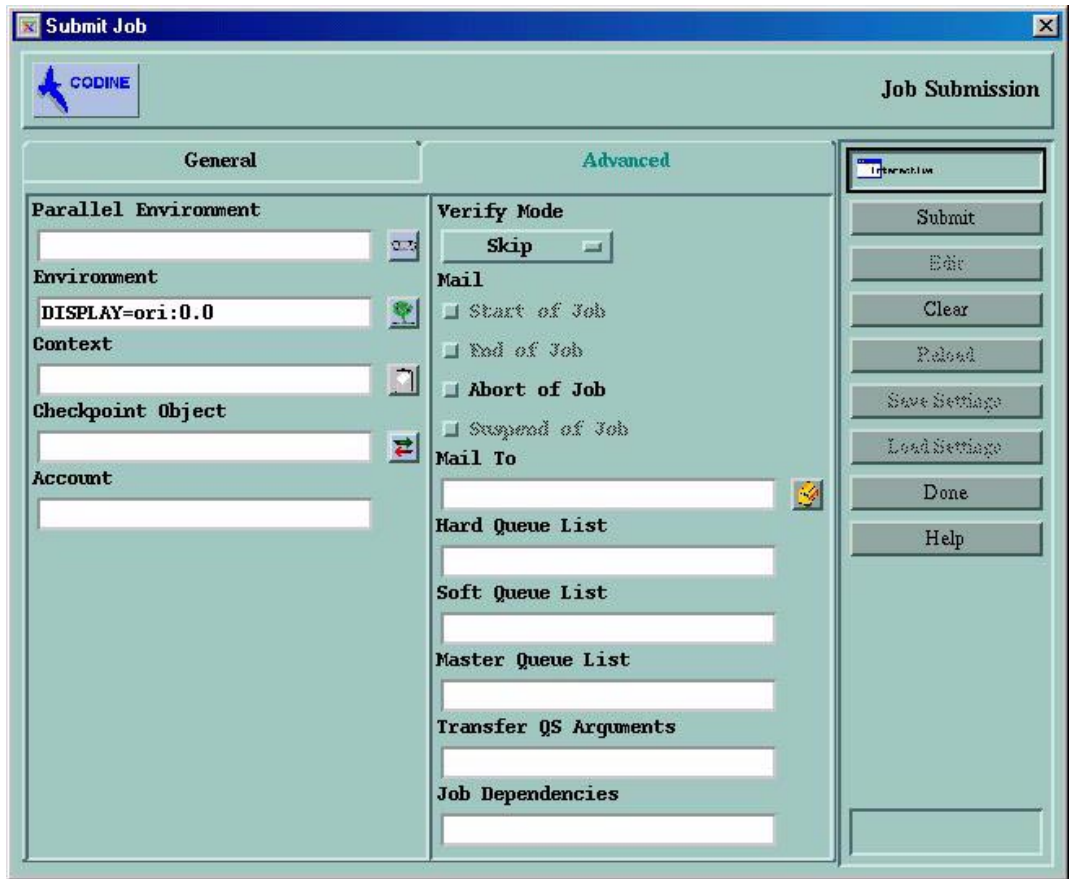


FIGURE 3-16 Interactive Job Submission dialogue Advanced

Submitting Interactive Jobs with qsh

Qsh is very similar to qsub and supports several of the qsub options as well as the additional switch `-display` to direct the display of the `xterm` to be invoked (please refer to the qsh manual page in the *Sun Grid Engine Reference Manual* for details).

The following command will start a `xterm` on any available Sun Solaris 64bit operating system host.

```
% qsh -l arch=solaris64
```

Submitting Interactive Jobs with `qlogin`

The `qlogin` command can be used from any terminal or terminal emulation to initiate an interactive session under the control of Sun Grid Engine. The following command will locate a low loaded host with Star-CD license available and with at least one queue providing a minimum of 6 hours hard CPU time limit.

```
% qlogin -l star-cd=1,h_cpu=6:0:0
```

Note – Depending on the remote login facility configured to be used by Sun Grid Engine you may be forced to enter your user name and/or password at a login prompt.

Transparent Remote Execution

Sun Grid Engine provides a set of closely related facilities supporting transparent remote execution of certain computational tasks. The core tool for this functionality is the `qrsh` command described in section “Remote Execution with `qrsh`” on page 204. Building on top of `qrsh`, two high level facilities - `qtcsh` and `qmake` - allow the transparent distribution of implicit computational tasks via Sun Grid Engine, thereby enhancing the standard Unix facilities `make` and `csch`. `Qtcsh` is explained in section “Transparent Job Distribution with `qtcsh`” on page 206 and `qmake` is described in section “Parallel Makefile Processing with `qmake`” on page 208.

Remote Execution with `qrsh`

`Qrsh` is built around the standard `rsh` facility (see the information provided in `<codine_root>/3rd_party` for details on the involvement of `rsh`) and can be used for various purposes:

- to provide remote execution of interactive applications via Sun Grid Engine comparable to the standard Unix facility `rsh` (also called `remsh` for HP-UX).
- to offer interactive login session capabilities via Sun Grid Engine similar to the standard Unix facility `rlogin` (note that `qlogin` is still required as a Sun Grid Engine representation of the Unix `telnet` facility).
- to allow for the submission of batch jobs which, upon execution, support terminal I/O (standard/error output and standard input) and terminal control.

- to offer a means for submitting a standalone program not embedded in a shell-script.
- to provide a batch job submission client which remains active while the job is pending or executing and which only finishes if the job has completed or has been cancelled.
- to allow for the Sun Grid Engine-controlled remote execution of job tasks (such as the concurrent tasks of a parallel job) within the framework of the dispersed resources allocated by parallel jobs (see section “Tight Integration of PEs and Sun Grid Engine” on page 152 of the *Sun Grid Engine Installation and Administration Guide*).

By virtue of all these capabilities, `qrsh` is the major enabling infrastructure for the implementation of the `qtcsh` and the `qmake` facilities as well as for the so called tight integration of Sun Grid Engine with parallel environments such as MPI or PVM.

Qrsh Usage

The general form of the `qrsh` command is:

```
% qrsh [options] program|shell-script [arguments] \
      [> stdout_file] [>&2 stderr_file] [< stdin_file]
```

`Qrsh` understands almost all options of `qsub` and provides only a few additional ones. These are:

- `-now yes|no`
controls whether the job is scheduled immediately and rejected if no appropriate resources are available, as usually desired for an interactive job – hence it is the default, or whether the job is queued like a batch job, if it cannot be started at submission time.
- `-inherit`
`qrsh` does not go through the Sun Grid Engine scheduling process to start a job-task, but it assumes that it is embedded inside the context of a parallel job which already has allocated suitable resources on the designated remote execution host. This form of `qrsh` commonly is used within `qmake` and within a tight parallel environment integration. The default is not to inherit external job resources.
- `-verbose`
presents output on the scheduling process. Mainly intended for debugging purposes and therefore switched off per default.

Transparent Job Distribution with `qtcsh`

`Qtcsh` is a fully compatible replacement for the widely known and used Unix C-Shell (`csch`) derivative `tcsh` (`qmake` is built around `tcsh` - see the information provided in `<codine_root>/3rd_party` for details on the involvement of `tcsh`). It provides a command-shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts via Sun Grid Engine. Which applications are to be executed remotely and which requirements apply for the selection of an execution host is defined in configuration files called `.qtask`.

Transparent to the user, such applications are submitted for execution to Sun Grid Engine via the `qrsh` facility. Since `qrsh` provides standard output, error output and standard input handling as well as terminal control connection to the remotely executing application, there are only three noticeable differences between executing such an application remotely as opposed to executing it on the same host as the shell:

1. The remote host may be much better suited (more powerful, lower loaded, required hard/software resources installed) than the local host, which may not allow execution of the application at all. This is a desired difference, of course.
2. There will be a small delay incurred by the remote startup of the jobs and by their handling through Sun Grid Engine.
3. Administrators can restrict the usage of resources through interactive jobs (`qrsh`) and thus through `qtcsh`. If not enough suitable resources are available for an application to be started via the `qrsh` facility or if all suitable systems are overloaded, the implicit `qrsh` submission will fail and a corresponding error message will be returned ("not enough resources ... try later").

In addition to the *standard* use, `qtcsh` is a suitable platform for third party code and tool integration. Using `qtcsh` in its single-application execution form `qtcsh -c appl_name` inside integration environments presents a persistent interface that almost never has to be changed. All the required application, tool, integration, site and even user specific configurations are contained in appropriately defined `.qtask` files. A further advantage is that this interface can be used from within shell scripts of any type, C programs and even Java applications.

`Qtcsh` Usage

Invocation of `qtcsh` is exactly the same as for `tcsh`. `Qtcsh` extends `tcsh` in providing support for the `.qtask` file and by offering a set of specialized shell built-in modes.

The `.qtask` file is defined as follows: Each line in the file has the format:

```
% [!]appl_name qrsh_options
```

The optional leading exclamation mark “!” defines the precedence between conflicting definitions in a cluster global `.qtask` file and the personal `.qtask` file of the `qtcs` user. If the exclamation mark is missing in the cluster global file, an eventually conflicting definition in the user file will overrule. If the exclamation mark is in the cluster global file, the corresponding definition cannot be overwritten.

The rest of the line specifies the name of the application which, when typed on a command line in a `qtcs`, will be submitted to Sun Grid Engine for remote execution, and the options to the `qrsh` facility, which will be used and which define resource requirements for the application.

Note – The application name must appear in the command line exactly like defined in the `.qtask` file. If it is prefixed with an absolute or relative directory specification it is assumed that a local binary is addressed and no remote execution is intended.

Note – Csh aliases, however, are expanded before a comparison with the application names is performed. The applications intended for remote execution can also appear anywhere in a `qtcs` command line, in particular before or after standard I/O redirections.

Hence, the following examples are valid and meaningful syntax:

```
# .qtask file
netscape -v DISPLAY=myhost:0
grep -l h=filesurfer
```

Given this .qtask file, the following qtcsh command lines:

```
netscape
~/mybin/netscape
cat very_big_file | grep pattern | sort | uniq
```

will implicitly result in:

```
qrsh -v DISPLAY=myhost:0 netscape
~/mybin/netscape
cat very_big_file | qrsh -l h=filesurfer grep
```

Qtcsh can operate in different modes influenced by switches where each of them can be on or off:

- Local or remote execution of commands (remote is default).
- Immediate or batch remote execution (immediate is default).
- Verbose or non-verbose output (non-verbose is default).

The setting of these modes can be changed using option arguments of qtcsh at start time or with the shell builtin command qrshmode at runtime. See the qtcsh manual page in the *Sun Grid Engine Reference Manual* for more information.

Parallel Makefile Processing with qmake

Qmake is a replacement for the standard Unix make facility. It extends make by its ability to distribute independent make steps across a cluster of suitable machines. Qmake is built around the popular GNU-make facility gmake. See the information provided in <codine_root>/3rd_party for details on the involvement of gmake.

To ensure that a complex distributed make process can run to completion, qmake first allocates the required resources in an analogous form like a parallel job. Qmake then manages this set of resources without further interaction with the Sun Grid Engine scheduling. It distributes make steps as resources are or become available via the qrsh facility with the -inherit option enabled.

Since `qrsh` provides standard output, error output and standard input handling as well as terminal control connection to the remotely executing make step, there are only three noticeable differences between executing a make procedure locally or using `qmake`:

1. Provided that the individual make steps have a certain duration and that there are enough independent make steps to be processed, the parallelization of the make process will be sped up significantly. This is a desired difference, of course.
2. With each make step to be started up remotely there will be an implied small overhead caused by `qrsh` and the remote execution as such.
3. To take advantage of the make step distribution of `qmake`, the user has to specify as a minimum the degree of parallelization, i.e. the number of concurrently executable make steps. In addition, the user can specify the resource characteristics required by the make steps, such as available software licenses, machine architecture, memory or CPU-time requirements.

The most common use in general of make certainly is the compilation of complex software packages. This may not be the major application for `qmake`, however. Program files are often quite small (as a matter of good programming practice) and hence compilation of a single program file, which is a single make step, often only takes a few seconds. Furthermore, compilation usually implies a lot of file access (nested include files) which may not be accelerated if done for multiple make steps in parallel, because the file server can become the bottleneck effectively serializing all the file access. So a satisfactory speed-up of the compilation process sometimes cannot be expected.

Other potential applications of `qmake` are more appropriate. An example is the steering of the interdependencies and the workflow of complex analysis tasks through make-files. This is common in some areas, such as EDA, and each make step in such environments typically is a simulation or data analysis operation with non-negligible resource and computation time requirements. A considerable speed-up can be achieved in such cases.

Qmake Usage

The command-line syntax of `qmake` looks very similar to the one of `qrsh`:

```
% qmake [-pe pe_name pe_range] [further options] \  
-- [gnu-make-options] [target]
```

Note – The `-inherit` option is also supported by `qmake` as described further down below.

Specific attention has to be paid on the usage of the `-pe` option and its relation to the `qmake -j` option. Both options can be used to express the amount of parallelism to be achieved. The difference is that `qmake` provides no possibility with `-j` to specify something like a parallel environment to use. Hence, `qmake` makes the assumption, that a default environment for parallel makes is configured which is called `make`. Furthermore, `qmake's -j` allows no specification of a range, but only for a single number. `Qmake` will interpret the number given with `-j` as a range of `1-<given_number>`. As opposed to this, `-pe` permits the detailed specification of all these parameters. Consequently, the following command-line examples are identical

```
% qmake -- -j 10
% qmake -pe make 1-10 --
```

while the following command-lines cannot be expressed via the `-j` option:

```
% qmake -pe make 5-10,16
% qmake -pe mpi 1-99999
```

Apart from the syntax, `qmake` supports two modes of invocation: interactively from the command-line (without `-inherit`) or within a batch job (with `-inherit`). These two modes initiate a different sequence of actions:

1. interactive – when `qmake` is invoked on the command-line, the `make` process as such is implicitly submitted to Sun Grid Engine via `qcrsh` taking the resource requirements specified in the `qmake` command-line into account. Sun Grid Engine then selects a *master machine* for the execution of the parallel job associated with the parallel `make` job and starts the `make` procedure there. This is necessary, because the `make` process can be architecture dependent and the required architecture is specified in the `qmake` command-line. The `qmake` process on the master machine then delegates execution of individual `make` steps to the other hosts which have been allocated by Sun Grid Engine for the job and which are passed to `qmake` via the parallel environment hosts file.
2. batch – in this case, `qmake` appears inside a batch script with the `-inherit` option (if the `-inherit` option was not present, a new job would be spawned as described for the first case above). This results in `qmake` making use of the resources already allocated to the job into which `qmake` is embedded. It will use `qcrsh -inherit` directly to start `make` steps. When calling `qmake` in batch mode, the specification of resource requirements or `-pe` and `-j` options is ignored.

Note – Also single CPU jobs have to request a parallel environment (`qmake -pe make 1 --`). If no parallel execution is required, call `qmake` with `gmake` command-line syntax (without Sun Grid Engine options and “--”), it will behave like `gmake`.

Please refer to the `qmake` manual page in the *Sun Grid Engine Reference Manual* for further detail on `qmake`.

Checkpointing Jobs

User Level Checkpointing

Lots of application programs, especially those, which normally consume considerable CPU time, have implemented checkpointing and restart mechanisms to increase fault tolerance. Status information and important parts of the processed data are repeatedly written to one or more files at certain stages of the algorithm. These files (called restart files) can be processed if the application is aborted and restarted at a later time and a consistent state can be reached, comparable to the situation just before the checkpoint. As the user mostly has to deal with the restart files, e.g. in order to move them to a proper location, this kind of checkpointing is called *user level* checkpointing.

For application programs which do not have an integrated (user level) checkpointing an alternative can be to use a so called *checkpointing library* which can be provided by the public domain (see the *Condor* project of the University of Wisconsin for example) or by some hardware vendors. Re-linking an application with such a library installs a checkpointing mechanism in the application without requiring source code changes.

Kernel Level Checkpointing

Some operating systems provide checkpointing support inside the operating system kernel. No preparations in the application programs and no re-linking of the application is necessary in this case. Kernel level checkpointing is usually applicable for single processes as well as for complete process hierarchies. I.e., a hierarchy of interdependent processes can be checkpointed and restarted at any time. Usually both, a user command and a C-library interface are available to initiate a checkpoint.

Sun Grid Engine supports operating system checkpointing if available. Please refer to the Sun Grid Engine Release Notes for information on the currently supported kernel level checkpointing facilities.

Migration of Checkpointing Jobs

Checkpointing jobs are interruptible at any time, since their restart capability ensures that only few work already done must be repeated. This ability is used to build Sun Grid Engine's migration and dynamic load balancing mechanism. If requested, checkpointing Sun Grid Engine jobs are aborted on demand and migrated to other machines in the Sun Grid Engine pool thus averaging the load in the cluster in a dynamic fashion. Checkpointing jobs are aborted and migrated for the following reasons:

- The executing machine exceeds a load value configured to force a migration (`migr_load_thresholds` - see the `queue_conf` manual page in the *Sun Grid Engine Reference Manual*).
- The executing queue or the job is suspended, either explicitly by `qmod` or `qmon` or automatically if a suspend threshold for the queue (see section "Configuring Load and Suspend Thresholds" on page 79 of the *Sun Grid Engine Installation and Administration Guide*) has been exceeded and if the checkpoint occasion specification for the job (see section "Submit/Monitor/Delete a Checkpointing Job" on page 213) includes the suspension case.

You can identify a job which is about to migrate by the state `m` for migrating in the `qstat` output. A migrating job moves back to `cod_qmaster` and is subsequently dispatched to another suitable queue if any is available.

Composing a Checkpointing Job Script

Shell scripts for kernel level checkpointing show no difference from regular shell scripts.

Shell scripts for user level checkpointing jobs differ from regular Sun Grid Engine batch scripts only in their ability to properly handle the case if they get restarted. The environment variable `RESTARTED` is set for checkpointing jobs which are restarted. It can be used to skip over sections of the job script which should be executed during the initial invocation only.

Thus, a transparently checkpointing job script may look similar to the one given below:

Example Script File

```
#!/bin/sh
#Force /bin/sh in Sun Grid Engine
#$ -S /bin/sh
# Test if restarted/migrated
if [ $RESTARTED = 0 ]; then
    # 0 = not restarted
    # Parts to be executed only during the first
    # start go in here
    set_up_grid
fi
# Start the checkpointing executable
fem
#End of scriptfile
```

It is important to note that the job script is restarted from the beginning if a user level checkpointing job is migrated. The user is responsible for directing the program flow of the shell-script to the location where the job was interrupted and thus skipping those lines in the script which are critical to be executed more than once.

Note – Kernel level checkpointing jobs are interruptible at any point of time and also the embracing shell script is restarted exactly from the point where the last checkpoint occurred. Therefore, the `RESTARTED` environment variable are of no relevance for kernel level checkpointing jobs.

Submit/Monitor/Delete a Checkpointing Job

Submitting a checkpointing job works the same way as for regular batch scripts except for the `qsub -ckpt` and `-c` switches, which request a checkpointing mechanism and define the occasions at which checkpoints have to be generated for the job. The `-ckpt` option takes one argument which is the name of the checkpointing environment (see section “Checkpointing Support” on page 140 in the

Sun Grid Engine Installation and Administration Guide) to be used. The `-c` option is not mandatory and also takes one argument. It can be used to overwrite the definitions of the `when` parameter in the checkpointing environment configuration (see the `checkpoint` manual page in the *Sun Grid Engine Reference Manual* for details).

The argument to the `-c` option can be one of the following one letter selection (or any combination thereof) or a time value alternatively:

- `n`
no checkpoint is performed. This has highest precedence
- `s`
A checkpoint is only generated if the `cod_execd` on the jobs host is shut down.
- `m`
Generate checkpoint at minimum CPU interval defined in the corresponding queue configuration (see the `min_cpu_interval` parameter in the `queue_conf` manual page).
- `x`
A checkpoint is generated if the job gets suspended.
- `interval`
Generate checkpoint in the given interval but not more frequently than defined by `min_cpu_interval` (see above). The time value has to be specified as `hh:mm:ss` (two digit hours, minutes and seconds separated by colon signs).

The monitoring of checkpointing jobs just differs from regular jobs by the fact, that these jobs may migrate from time to time (signified by state `m` for migrating in the output of `qstat`, see above) and, therefore, are not bound to a single queue. However, the unique job identification number stays the same as well as the job name.

Deleting checkpointing jobs works just the same way as described in section “Controlling Sun Grid Engine Jobs from the Command-line” on page 230.

Submit a Checkpointing Job with `qmon`

Submission of checkpointing jobs via `qmon` is identical to the submission of regular batch jobs with the addition of specifying an appropriate checkpointing environment. As explained in “Submitting Jobs with `qmon` (Advanced Example)” on page 181 the Job Submission dialogue provides an input window for the checkpointing environment associated with a job. Aside to the input window there is an icon button, which opens the selection dialogue displayed in figure 3-17 on page 215. You can select a suitable checkpoint environment from the list of available

ones with it. Please ask your system administrator for information on the properties of the checkpointing environments installed at your site or refer to section “Checkpointing Support” on page 140.



FIGURE 3-17 Checkpoint Object Selection

File System Requirements

When a checkpointing library based user level or kernel level checkpoint is written, a complete image of the virtual memory the process or job to be checkpointed covers needs to be dumped. Sufficient disk space must be available for this purpose. If the checkpointing environment configuration parameter `ckpt_dir` is set the checkpoint information is dumped to a job private location under `ckpt_dir`. If `ckpt_dir` is set to `NONE`, the directory in which the checkpointing job was started is used. Please refer to the manual page `checkpoint` in the *Sun Grid Engine Reference Manual* for detailed information about the checkpointing environment configuration.

Note – You should start a checkpointing job with the `qsub -cwd` script if `ckpt_dir` is set to `NONE`.

An additional requirement concerning the way how the file systems are organized is caused by the fact, that the checkpointing files and the restart files must be visible on all machines in order to successfully migrate and restart jobs. Thus NFS or a similar file system is required. Ask your cluster administration, if this requirement is met for your site.

If your site does not run NFS or if it is not desirable to use it for some reason, you should be able to transfer the restart files explicitly at the beginning of your shell script (e.g. via `rcp` or `ftp`) in the case of user level checkpointing jobs.

Monitoring and Controlling Sun Grid Engine Jobs

In principle, there are three ways to monitor submitted jobs: with the Sun Grid Engine graphical user's interface `qmon`, from the command-line with the `qstat` command or by electronic mail.

Monitoring and Controlling Jobs with `qmon`

The Sun Grid Engine graphical user's interface `qmon` provides a dialogue specifically designed for controlling jobs. The `Job Control` dialogue is opened by pushing the `Job Control` icon button in the `qmon` main menu.

The general purpose of this dialogue is to provide the means to monitor all running, pending and a configurable number of finished jobs known to the system or parts thereof. The dialogue can also be used to manipulate jobs, i.e. to change their priority, to suspend, resume and to cancel them. Three list environments are displayed, one for the running jobs, another for the pending jobs waiting to be dispatched to an appropriate resource and the third for recently finished jobs. You can select between the three list environments via clicking to the corresponding tab labels at the top of the screen.

In its default form (see figure 3-18 on page 220) it displays the columns `JobId`, `Priority`, `JobName` and `Queue` for each running and pending job. The set of information displayed can be configured with a customization dialogue (see figure 3-18 on page 220), which is opened upon pushing the `Customize` button in the `Job Control` dialogue. With the customization dialogue it is possible to select further entries of the Sun Grid Engine job object to be displayed and to filter the jobs of interest. The example on page 220 selects the additional fields `MailTo` and `Submit Time`. The `Job Control` dialogue displayed in figure 3-18 on page 220 depicts the enhanced look after the customization has been applied in case of the `Finished Jobs` list. The example of the filtering facility in figure 3-21 on page 223 selects only those jobs owned by *ferstl* which run or are suitable for architecture `solaris64`. The resulting `Job Control` dialogue showing `Pending Jobs` is displayed in figure 3-22 on page 224.

Note – The `Save` button the customize dialogue displayed on page 220, for example, stores the customizations into the file `.qmon_preferences` in the user's home directory and thus redefines the default appearance of the job control dialogue.

The Job Control dialogue in figure 3-22 on page 224 is also an example for how array jobs are displayed in `qmon`.

Jobs can be selected (for later operation) with the following mouse/key combinations:

- Clicking to a job with the left mouse button while the Control key is pressed starts a selection of multiple jobs.
- Clicking to another job with the left mouse button while the Shift key is pressed selects all jobs in between and including the job at the selection start and the current job.
- Clicking to a job with the left mouse button while the Control and the Shift key are pressed toggles the selection state of a single job.

The selected jobs can be suspended, resumed (unsuspended), deleted, held back (and released), re-prioritized and modified (Qalter) through the Corresponding buttons at the right side of the screen.

The actions `suspend`, `unsuspend`, `delete`, `hold`, `modify priority` and `modify job` may only be applied to a job by the job owner or by Sun Grid Engine managers and operators (see “Managers, Operators and Owners” on page 173). Only running jobs can be suspended/resumed and only pending jobs can be held back and modified (in priority as well as in other attributes).

Suspending a job means the equivalent to sending the signal `SIGSTOP` to the process group of the job with the UNIX `kill` command. I.e., the job is halted and does no longer consume CPU time. Unsuspending the job sends the signal `SIGCONT` thereby resuming the job (see the `kill` manual page of your system for more information on signalling processes).

Note – Suspension, unsuspension and deletion can be forced, i.e. registered with `cod_qmaster` without notification of the `cod_execd` controlling the job(s), in case the corresponding `cod_execd` is unreachable, e.g. due to network problems. Use the `Force` flag for this purpose.

If using the `Hold` button on a selected pending job, the `Set Hold` sub-dialogue is opened (see figure 3-18 on page 220). It allows to set and to reset user, system and operator holds. User holds can be set/reset by the job owner as well as Sun Grid Engine operators and managers. Operator holds can be set/reset by managers and operator and manager holds can be set/reset by managers only. As long as any hold

is assigned to a job it is not eligible for execution. An alternate way to set/reset holds are the `qalter`, `qhold` and `qrls` commands (see the corresponding manual pages in *Sun Grid Engine Reference Manual*).

If the `Priority` button is pressed another sub-dialogue is opened (figure 3-18 on page 220), which allows to enter the new priority of the selected pending jobs. The priority determines the order of the jobs in the pending jobs list and the order in which the pending jobs are displayed by the `Job Control` dialogue. Users can only set the priority in the range between 0 and -1024. Sun Grid Engine operators and managers can also increase the priority level up to the maximum of 1023 (see section “Job Priorities” on page 127 in the *Sun Grid Engine Installation and Administration Guide* for details about job priorities).

The `Qalter` button, when pressed for a pending job, opens the `Job Submission` screen described in “Submitting Sun Grid Engine Jobs” on page 175 with all the entries of the dialogue set corresponding to the attributes of the job as defined during submission. Those entries, which cannot be changed are set insensitive. The others may be edited and the changes are registered with Sun Grid Engine by pushing the `Qalter` button (a replacement for the `Submit` button) in the `Job Submission` dialogue.

The `Verify` flag in the `Job Submission` screen has a special meaning when used in the “`qalter`” mode. You can check pending jobs for their consistency and investigate why they have not been scheduled yet. You just have to select the desired consistency checking mode for the `Verify` flag and push the `Qalter` button. The system will display warnings on inconsistencies depending on the selected checking mode. Please refer to “Submitting Jobs with `qmon` (Advanced Example)” on page 181 and the `-w` option in the `qalter` manual page for further information.

Another method for checking why jobs are still pending is to select a job and click on the “Why ?” button of the `Job Control` dialogue. This will open the `Object Browser` dialogue and display a list of reasons which prevented the Sun Grid Engine scheduler from dispatching the job in its most recent pass. An example browser screen displaying such a message is shown in figure 3-25 on page 226.

Note – The “Why ?” button only delivers meaningful output if the scheduler configuration parameter `schedd_job_info` is set to true (see `sched_conf` in the *Sun Grid Engine Reference Manual*).

Note – The displayed scheduler information relates to the last scheduling interval. It may not be accurate anymore by the time you investigate for reasons why your job has not been scheduled.

The `Clear Error` button can be used to remove an error state from a selected pending job, which had been started in an earlier attempt, but failed due to a job dependent problem (e.g., insufficient permissions to write to the specified job output file).

Note – Error states are displayed using a red font in the pending jobs list and should only be removed after correcting the error condition, e.g., via `qalter`.

Note – Such error conditions are automatically reported via electronic mail, if the job requests to send e-mail in cases it is aborted (e.g. via the `qsub -m a` option).

To keep the information being displayed up-to-date, `qmon` uses a polling scheme to retrieve the status of the jobs from `cod_qmaster`. An update can be forced by pressing the `Refresh` button.

Finally, the button provides a link to the `qmon Job Submission` dialogue (see figure 3-8 on page 181 for example).

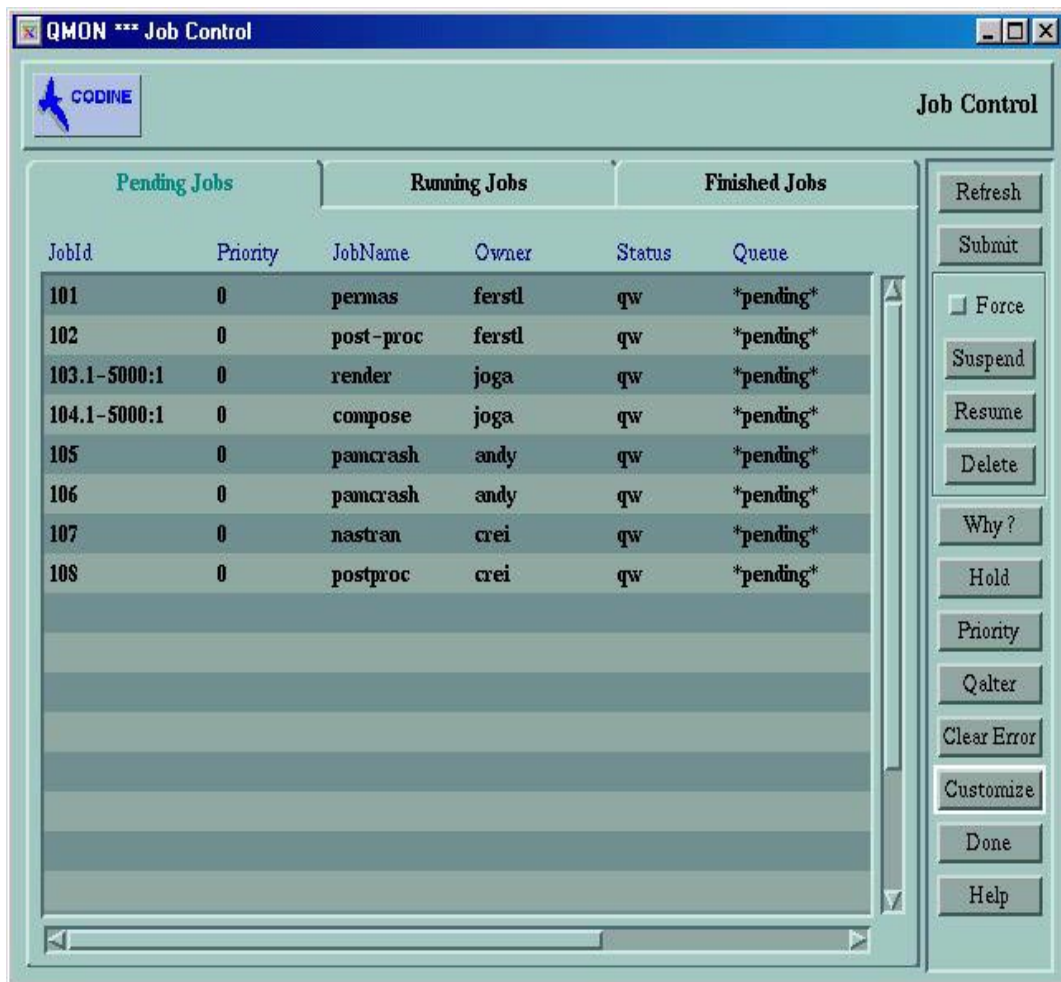


FIGURE 3-18 Job Control dialogue - standard form

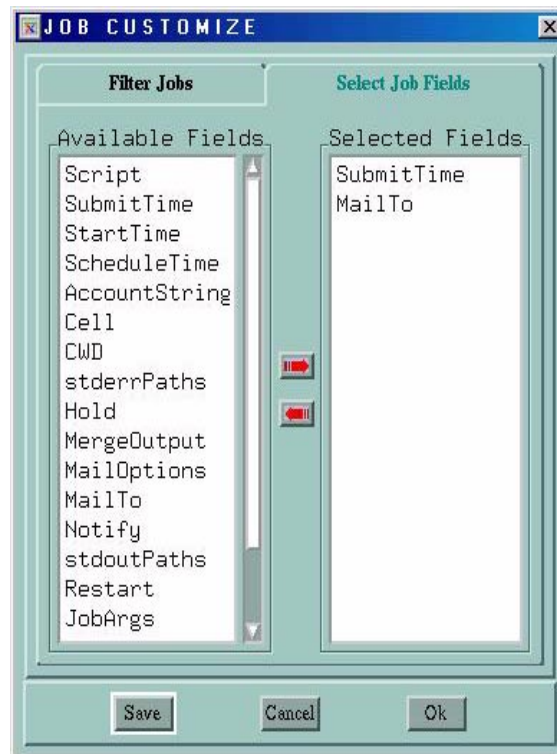


FIGURE 3-19 Job Control customization

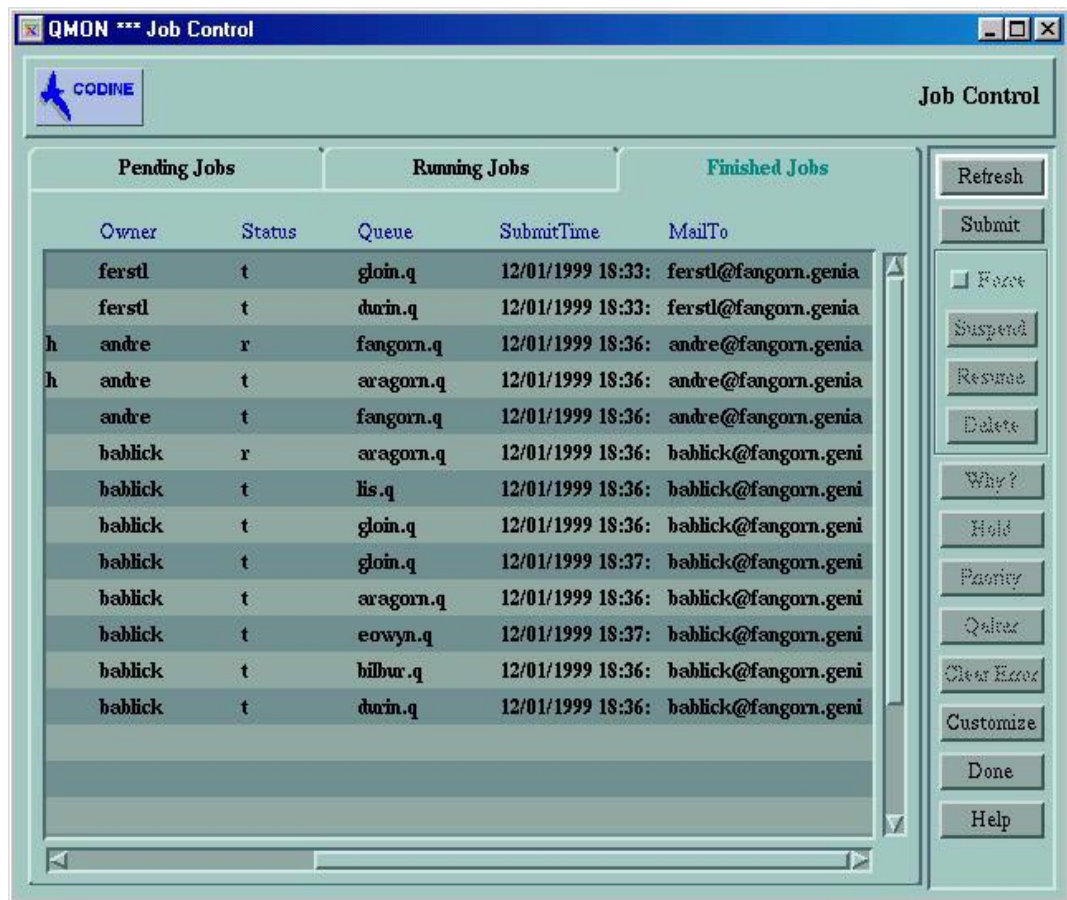


FIGURE 3-20 Job Control dialogue Finished Jobs - enhanced

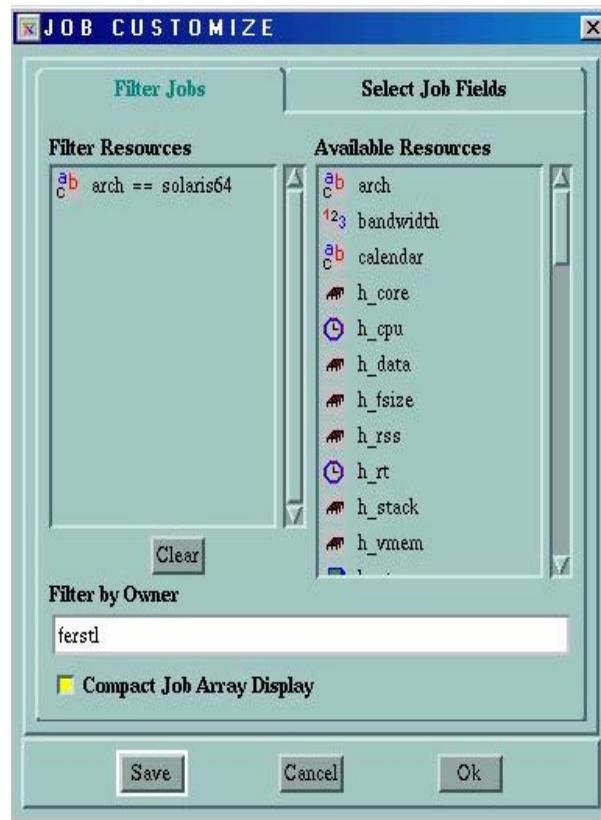


FIGURE 3-21 Job Control filtering



FIGURE 3-22 Job Control dialogue - after filtering



FIGURE 3-23 Job Control holds

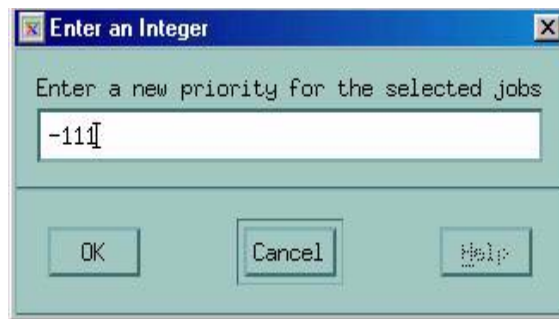


FIGURE 3-24 Job Control priority definition

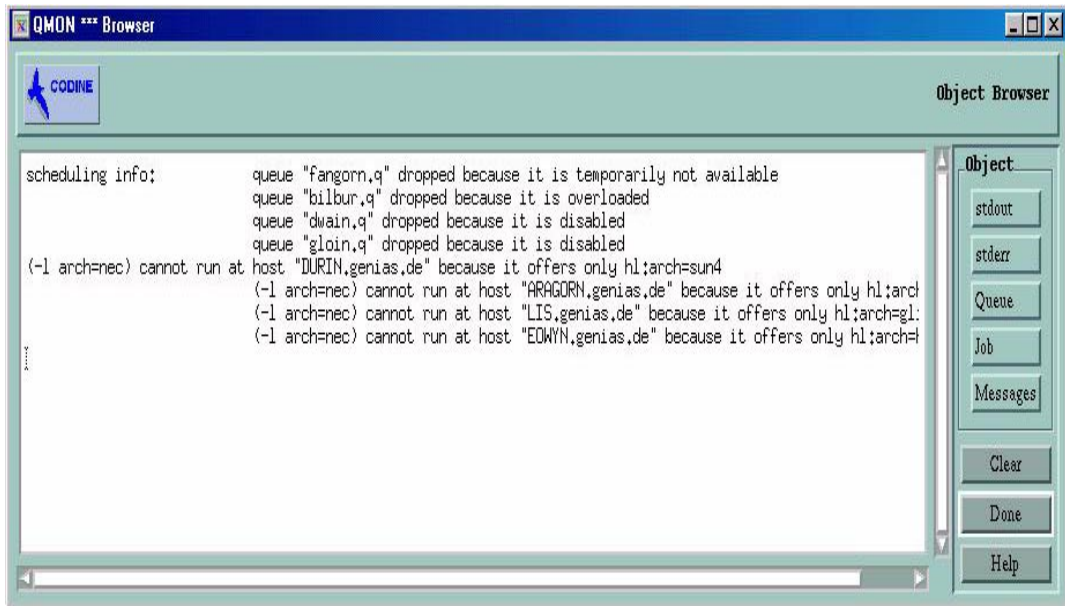


FIGURE 3-25 Browser displaying scheduling information

Additional Information with the qmon Object Browser

The qmon Object Browser can be used to quickly retrieve additional information on Sun Grid Engine jobs without a need to customize the Job Control dialogue as explained in section "Monitoring and Controlling Jobs with qmon" on page 216.

The Object Browser is opened upon pushing the Browser icon button in the qmon main menu. The browser displays information about Sun Grid Engine jobs if the Job button in the browser is selected and if the mouse pointer is moved over a job's line in the Job Control dialogue (see figure 3-18 on page 220 for example).

The browser screen in figure 3-26 on page 227 gives an example of the information displayed in such a situation.

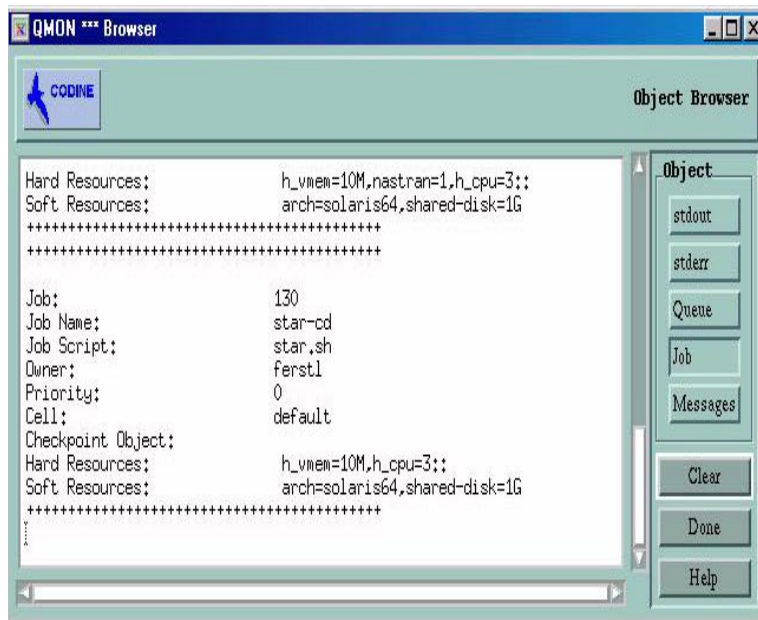


FIGURE 3-26 Object Browser - job

Monitoring with `qstat`

Submitted jobs can also be monitored with the Sun Grid Engine `qstat` command. There are two basic forms of the `qstat` command available:

```
% qstat
% qstat -f
```

The first form provides an overview on the submitted jobs only (see table 3-3 on page 229). The second form includes information on the currently configured queues in addition (see table 3-4 on page 229).

In the first form, a header line indicates the meaning of the columns. The purpose of most of the columns should be self-explanatory. The `state` column, however, contains single character codes with the following meaning: `r` for running, `s` for suspended, `q` for queued and `w` for waiting (see the `qstat` manual page in the *Sun Grid Engine Reference Manual* for a detailed explanation of the `qstat` output format).

The second form is divided into two sections, the first displaying the status of all available queues, the second (entitled with the `- PENDING JOBS - . . .` separator) shows the status of the `cod_qmaster` job spool area. The first line of the queue section defines the meaning of the columns with respect to the enlisted queues. The queues are separated by horizontal rules. If jobs run in a queue they are printed below the associated queue in the same format as in the `qstat` command in its first form. The pending jobs in the second output section are also printed as in `qstat`'s first form.

The following columns of the queue description require some explanation:

- `qtype`

The queue type - one of B(atch), I(nteractive), P(arallel) and C(heckpointing) or combinations thereof or alternatively T(ransfer).

- `used/free`

The count of used/free job slots in the queue.

- `states`

The state of the queue - one of u(nknown), a(larm), s(uspended), d(isabled), E(rror) or combinations thereof.

Again, the `qstat` manual page contains a more detailed description of the `qstat` output format.

Various additional options to the `qstat` command enhance the functionality in both versions. The `-r` option can be used to display the resource requirements of submitted jobs. Furthermore the output may be restricted to a certain user, to a specific queue and the `-l` option may be used to specify resource requirements as described in section "Resource Requirement Definition" on page 191 for the `qsub` command. If resource requirements are used, only those queues (and the jobs running in these queues) are displayed which match the resource requirement specification in the `qstat` command-line.

TABLE 3-3 qstat example output

| job-ID | prior | name | user | state | submit/start at | queue | function |
|--------|-------|-----------|---------|-------|-------------------|---------|----------|
| 231 | 0 | hydra | craig | r | 07/13/96 20:27:15 | durin.q | MASTER |
| 232 | 0 | compile | penny | r | 07/13/96 20:30:40 | durin.q | MASTER |
| 230 | 0 | blackhole | don | r | 07/13/96 20:26:10 | dwain.q | MASTER |
| 233 | 0 | mac | elaine | r | 07/13/96 20:30:40 | dwain.q | MASTER |
| 234 | 0 | golf | shannon | r | 07/13/96 20:31:44 | dwain.q | MASTER |
| 236 | 5 | word | elaine | qw | 07/13/96 20:32:07 | | |
| 235 | 0 | andrun | penny | qw | 07/13/96 20:31:43 | | |

TABLE 3-4 qstat -f example output

| queue | name | qtype | used/free | load_avg | arch | states |
|--|------|-----------|-----------|----------|-------------------|--------|
| dq | | BIP | 0/1 | 99.99 | sun4 | au |
| durin.q | | BIP | 2/2 | 0.36 | sun4 | |
| 231 | 0 | hydra | craig | r | 07/13/96 20:27:15 | MASTER |
| 232 | 0 | compile | penny | r | 07/13/96 20:30:40 | MASTER |
| dwain.q | | BIP | 3/3 | 0.36 | sun4 | |
| 230 | 0 | blackhole | don | r | 07/13/96 20:26:10 | MASTER |
| 233 | 0 | mac | elaine | r | 07/13/96 20:30:40 | MASTER |
| 234 | 0 | golf | shannon | r | 07/13/96 20:31:44 | MASTER |
| fq | | BIP | 0/3 | 0.36 | sun4 | |
| ##### | | | | | | |
| - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - | | | | | | |
| ##### | | | | | | |
| 236 | 5 | word | elaine | qw | 07/13/96 20:32:07 | |
| 235 | 0 | andrun | penny | qw | 07/13/96 20:31:43 | |

Monitoring by Electronic Mail

The `qsub -m` switch requests electronic mail to be sent to the user submitting a job or to the email address(es) specified by the `-M` flag if certain events occur (see the `qsub` manual page for a description of the flags). An argument to the `-m` option specifies the events. The following selections are available:

- `b`

Mail is sent at the beginning of the job.

- `e`

Mail is sent at the end of the job.

- `a`

Mail is sent when the job is aborted (e.g. by a `qdel` command).

- `s`

Mail is sent when the job is suspended.

- `n`

No mail is sent (the default).

Multiple of these options may be selected with a single `-m` option in a comma separated list.

The same mail events can be configured by help of the `qmon` Job Submission dialogue, see section “Submitting Jobs with `qmon` (Advanced Example)” on page 181.

Controlling Sun Grid Engine Jobs from the Command-line

The section “Monitoring and Controlling Jobs with `qmon`” on page 216 explains how Sun Grid Engine jobs can be deleted, suspended and resumed with the Sun Grid Engine graphical user’s interface `qmon`.

From the command-line, the `qdel` command can be used to cancel Sun Grid Engine jobs, regardless whether they are running or spooled. The `qmod` command provides the means to suspend and unsuspend (resume) jobs already running.

For both commands, you will need to know the job identification number, which is displayed in response to a successful `qsub` command. If you forget the number it can be retrieved via `qstat` (see section “Monitoring with `qstat`” on page 227).

Included below are several examples for both commands:

```
% qdel job_id
% qdel -f job_id1, job_id2
% qmod -s job_id
% qmod -us -f job_id1, job_id2
% qmon -s job_id.task_id_range
```

In order to delete, suspend or unsuspend a job you must be either the owner of the job, a Sun Grid Engine manager or operator (see “Managers, Operators and Owners” on page 173).

For both commands the `-f` force option can be used to register a status change for the job(s) at `cod_qmaster` without contacting `cod_execd` in case `cod_execd` is unreachable, e.g. due to network problems. The `-f` option is intended for usage by the administrator. In case of `qdel`, however, users can be enabled to force deletion of their own jobs if the flag `ENABLE_FORCED_QDEL` in the cluster configuration `qmaster_params` entry is set (see the `sgc_conf` manual page in the *Sun Grid Engine Reference Manual* for more information).

Job Dependencies

The most convenient way to build a complex task often is to split the task into sub-tasks. In these cases sub-tasks depend on the successful completion of other sub-tasks before they can get started. An example is that a predecessor task produces an output file which has to be read and processed by a successor task.

Sun Grid Engine supports interdependent tasks with its job dependency facility. Jobs can be configured to depend on the successful completion of one or multiple other jobs. The facility is enforced by the `qsub -hold_jid` option. A list of jobs can be specified upon which the submitted job depends. The list of jobs can also contain subsets of array jobs. The submitted job will not be eligible for execution unless all jobs in the dependency list have completed successfully.

Controlling Queues

As already stated in section “Queues and Queue Properties” on page 165, the owners of queues have permission to suspend/unsuspend or disable/enable queues. This is desirable, if these users need certain machines from time to time for important work and if they are affected strongly by Sun Grid Engine jobs running in the background.

There are two ways to suspend or enable queues. The first, using the `qmon Queue Control` dialogue and the second utilizing the `qmod` command.

Controlling Queues with `qmon`

Clicking on the `Queue Control` icon button in the `qmon` main menu brings up the `Queue Control` dialogue. An example screen is displayed in “Queue Control dialogue” on page 233.

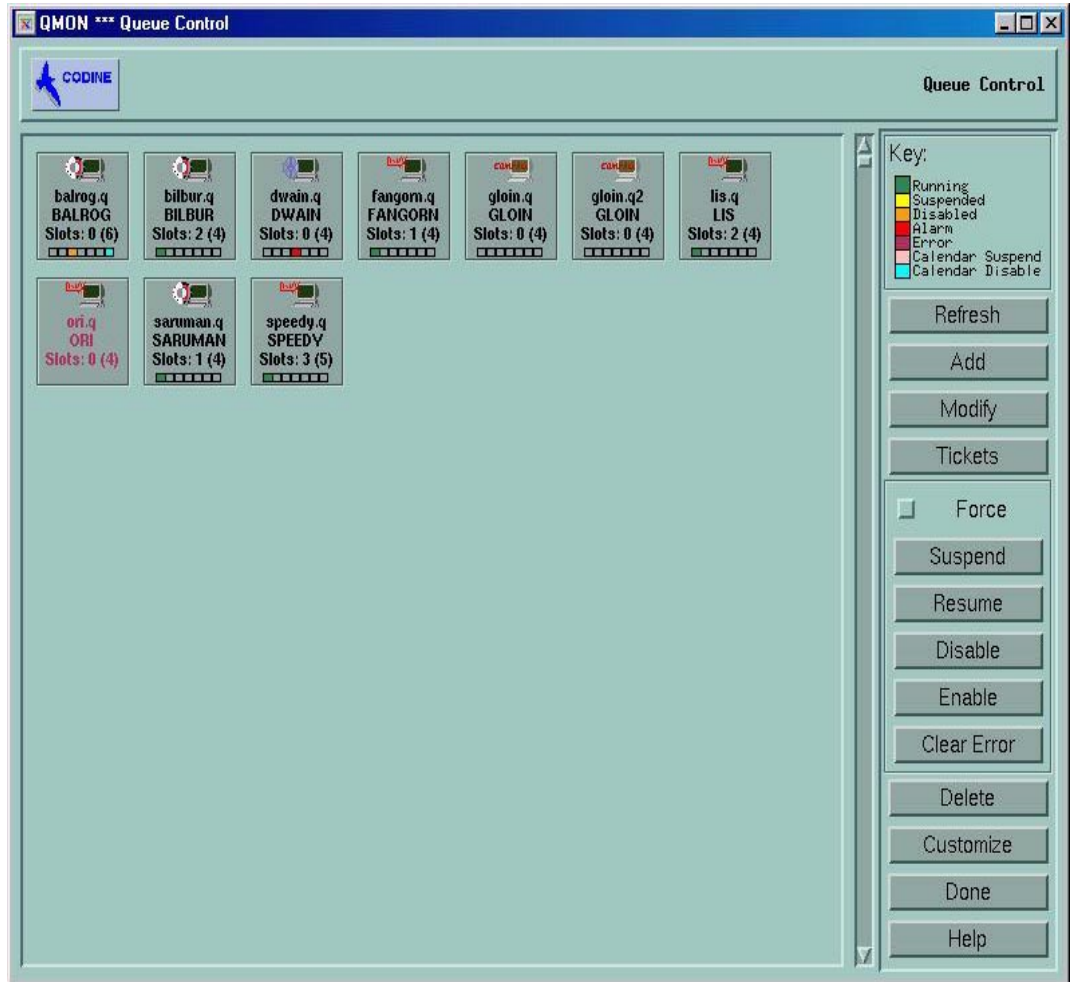


FIGURE 3-27 Queue Control dialogue

The purpose of the Queue Control dialogue is to provide a quick overview on the resources being available and on the activity in the cluster. It also provides the means to suspend/unsuspend and to disable/enable queues as well as to configure queues. Each icon being displayed represents a queue. If the main display area is empty, no queues are configured. Each queue icon is labelled with the queue name, the name of the host on which the queue resides and the number of job slots being occupied. If a `cod_execd` is running on the queue host and has already registered with `cod_qmaster` a picture on the queue icon indicates the queue host's operating system architecture and a color bar at the bottom of the icon informs about the status of the queue. A legend on the right side of the Queue Control dialogue displays the meaning of the colors.

For those queues, the user can retrieve the current attribute, load and resource consumption information for the queue and implicitly of the machine which hosts a queue by clicking to the queue icon with the left mouse button while the `Shift` key on the keyboard is pressed. This will pop-up an information screen similar to the one displayed in figure 3-28 on page 235 (see there for a detailed description).

Queues are selected by clicking with the left mouse on the button or into a rectangular area surrounding the queue icon buttons. The `Delete`, `Suspend/Unsuspend` or `Disable/Enable` buttons can be used to execute the corresponding operation on the selected queues. The suspend/unsuspend and disable/enable operation require notification of the corresponding `cod_execd`. If this is not possible (e.g. because the host is down) a `cod_qmaster` internal status change can be forced if the `Force` toggle button is switched on.

If a queue is suspended, the queue is closed for further jobs and the jobs already executing in the queue are suspended as explained in section “Monitoring and Controlling Jobs with `qmon`” on page 216. The queue and its jobs are resumed as soon as the queue is unsuspended.

Note – If a job in a suspended queue has been suspended explicitly in addition, it will not be resumed if the queue is unsuspended. It needs to be unsuspended explicitly again.

Queues which are disabled are closed, however, the jobs executing in those queues are allowed to continue. To disable a queue is commonly used to „drain“ a queue. After the queue is enabled, it is eligible for job execution again. No action on still executing jobs is performed.

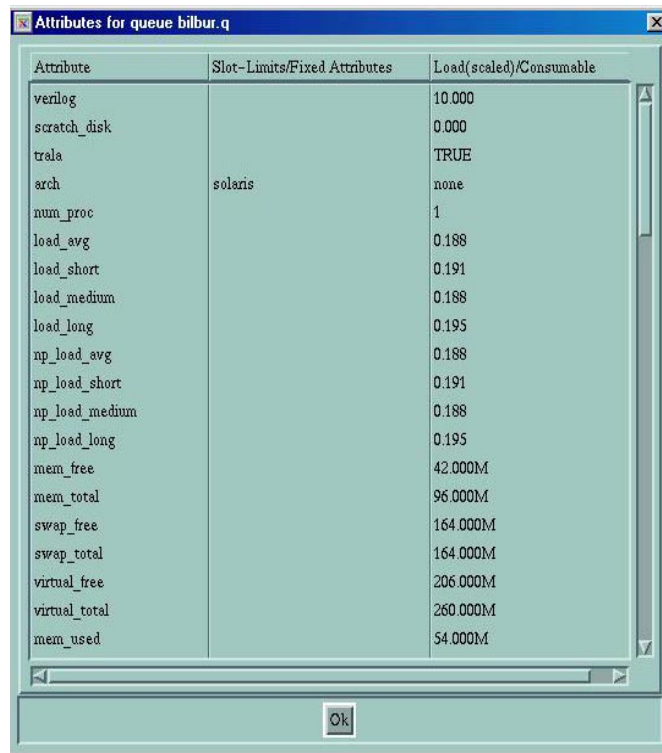
The suspend/unsuspend and disable/enable operations require queue owner or Sun Grid Engine manager or operator permission (see section “Managers, Operators and Owners” on page 173).

The information displayed in the `Queue Control` dialogue is update periodically. An update can be forced by pressing the `Refresh` button. The `Done` button closes the dialogue.

The `Customize` button allows you to select the queues to be displayed via a filter operation. The sample screen in figure 3-29 on page 236 shows the selection of only those queues which run on hosts belonging to architecture `osf4` (i.e Compaq Unix version 4). The `Save` button in the customize dialogue allows you to store your settings in the file `.qmon_preferences` in your home directory for standard reactivation on later invocations of `qmon`.

For the purpose of configuring queues a sub-dialogue is opened when pressing the `Add` or `Modify` button on the right side of the `Queue Control` screen (see section “Configuring Queues with `qmon`” on page 75 in the *Sun Grid Engine Installation and Administration Guide* for details).

In the following, a detailed description of the queue attribute screen displayed below is given:



The screenshot shows a window titled "Attributes for queue bilbur.q". It contains a table with three columns: "Attribute", "Slot-Limits/Fixed Attributes", and "Load(scaled)/Consumable". The table lists various system attributes and their values for the queue "bilbur.q".

| Attribute | Slot-Limits/Fixed Attributes | Load(scaled)/Consumable |
|----------------|------------------------------|-------------------------|
| verilog | | 10.000 |
| scratch_disk | | 0.000 |
| trala | | TRUE |
| arch | solaris | none |
| num_proc | | 1 |
| load_avg | | 0.188 |
| load_short | | 0.191 |
| load_medium | | 0.188 |
| load_long | | 0.195 |
| np_load_avg | | 0.188 |
| np_load_short | | 0.191 |
| np_load_medium | | 0.188 |
| np_load_long | | 0.195 |
| mem_free | | 42.000M |
| mem_total | | 96.000M |
| swap_free | | 164.000M |
| swap_total | | 164.000M |
| virtual_free | | 206.000M |
| virtual_total | | 260.000M |
| mem_used | | 54.000M |

FIGURE 3-28 Queue attribute display

All attributes attached to the queue (including those being inherited from the host or cluster) are listed in the **Attribute** column. The **Slot-Limits/Fixed Attributes** column shows values for those attributes being defined as per queue slot limits or as fixed complex attributes. The **Load (scaled) / Consumable** column informs about the reported (and if configured scaled) load parameters (see section “Load Parameters” on page 113 in the *Sun Grid Engine Installation and Administration Guide*) and about available resource capacities based on the Sun Grid Engine consumable resources facility (see section “Consumable Resources” on page 96).

Note – Load reports and consumable capacities may overwrite each other, if a load attribute is configured as a consumable resource. The minimum value of both, which is used in the job dispatching algorithm, is displayed.

Note – The displayed load and consumable values currently do not take into account load adjustment corrections as described in section “Execution Hosts” on page 62 of the *Sun Grid Engine Installation and Administration Guide*.



FIGURE 3-29 Queue Control customization

Controlling Queues with qmod

Section “Controlling Sun Grid Engine Jobs from the Command-line” on page 230 explained how the Sun Grid Engine command `qmod` can be used to suspend/unsuspend Sun Grid Engine jobs. However, the `qmod` command additionally provides the user with the means to suspend/unsuspend or disable/enable queues.

The following commands are examples how `qmod` is to be used for this purpose:

```
% qmod -s q_name
% qmod -us -f q_name1, q_name2
% qmod -d q_name
% qmod -e q_name1, q_name2, q_name3
```

The first two commands suspend or unsuspend queues, while the third and fourth command disable and enable queues. The second command uses the `qmod -f` option in addition to force registration of the status change in `cod_qmaster` in case the corresponding `cod_execd` is not reachable, e.g. due to network problems.

Note – Suspending/unsuspending as well as disabling/enabling queue requires queue owner, Sun Grid Engine manager or operator permission (see section “Managers, Operators and Owners” on page 173).

Note – You can use `qmod` commands with `crontab` or `at` jobs.

Customizing qmon

The look and feel of `qmon` is largely defined by a specifically designed resource file. Reasonable defaults are compiled-in and a sample resource file is available under `<codine_root>/qmon/Qmon`.

The cluster administration may install site specific defaults in standard locations such as `/usr/lib/X11/app-defaults/Qmon`, by including `qmon` specific resource definitions into the standard `.Xdefaults` or `.Xresources` files or by putting a site specific `Qmon` file to a location referenced by standard search paths such as `XAPPLRESDIR`. Please ask your administrator if any of the above is relevant in your case,

In addition, the user can configure personal preferences by either copying and modifying the `Qmon` file into the home directory (or to another location pointed to by the private `XAPPLRESDIR` search path) or by including the necessary resource definitions into the user’s private `.Xdefaults` or `.Xresources` files. A private `Qmon` resource file may also be installed via the `xrdb` command during operation or at start-up of the X11 environment, e.g. in a `.xinitrc` resource file.

Please refer to the comment lines in the sample `Qmon` file for detailed information on the possible customizations.

Another means of customizing qmon has been explained for the job and queue control customization dialogues shown in figure 3-18 on page 220 and in figure 3-29 on page 236. In both dialogues, the Save button can be used to store the filtering and display definitions configured with the customization dialogues to the file `.qmon_preferences` in the user's home directory. Upon being restarted, qmon will read this file and reactivate the previously defined behavior.

Reference Manual

Introduction

This document contains the manual pages as included in the Sun Grid Engine distribution.

Typographic Conventions

The following conventions are used in the Reference Manual

Sun Grid Engine as well as UNIX Commands which can be found in the following manual pages typeset in *emphasized* font. Command-line in- and output is printed in teletype font and newly introduced or defined terms are typeset in **boldface** font.

NAME

Sun Grid Engine Introduction – a facility for executing UNIX jobs on remote machines

DESCRIPTION

Sun Grid Engine is a facility for executing UNIX batch jobs (shell scripts) on a pool of cooperating workstations. Jobs are queued and executed remotely on workstations at times when those workstations would otherwise be idle or only lightly loaded. The work load is distributed among the workstations in the cluster corresponding to the load situation of each machine and the resource requirements of the jobs.

User level checkpointing programs are supported and a transparent checkpointing mechanism is provided (see *sge_ckpt(1)*). Checkpointing jobs migrate from workstation to workstation without user intervention on load demand. In addition to batch jobs, interactive jobs and parallel jobs can also be submitted to Sun Grid Engine. Sun Grid Engine also provides a mechanism for passing job requests over to arbitrary other queuing systems via the so called Queuing System Interface (QSI).

USER INTERFACE

The Sun Grid Engine user interface consists of several programs which are described separately.

qacct(1)

qacct extracts arbitrary accounting information from the cluster logfile.

qalter(1)

qalter changes the characteristics of already submitted jobs.

qconf(1)

qconf provides the user interface for configuring, modifying, deleting and querying queues and the cluster configuration.

qdel(1)

qdel provides the means for a user/operator/manager to cancel jobs.

qhold(1)

qhold holds back submitted jobs from execution.

qhost(1)

qhost displays status information about Sun Grid Engine execution hosts.

qlogin(1)

qlogin initiates a telnet or similar login session with automatic selection of a low loaded and suitable host.

qmake(1)

qmake is a replacement for the standard Unix *make* facility. It extends *make* by its ability to distribute independent *make* steps across a cluster of suitable machines.

qmod(1)

qmod allows the owner(s) of a queue to suspend and enable all queues associated with his machine (all currently active processes in this queue are also signaled) or to suspend and enable jobs executing in the owned queues.

qmon(1)

qmon provides a Motif command interface to all Sun Grid Engine functions. The status of all or a private selection of the configured queues is displayed on-line by changing colors at corresponding queue icons.

qresub(1)

qresub creates new jobs by copying currently running or pending jobs.

qrls(1)

qrls releases holds from jobs previously assigned to them e.g. via *qhold(1)* (see above).

qrsh(1)

qrsh can be used for various purposes such as providing remote execution of interactive applications via Sun Grid Engine comparable to the standard Unix facility *rsh*, to allow for the submission of batch jobs which, upon execution, support terminal I/O (standard/error output and standard input) and terminal control, to provide a batch job submission client which remains active until the job has finished or to allow for the Sun Grid Engine-controlled remote execution of the tasks of parallel jobs.

qselect(1)

qselect prints a list of queue names corresponding to specified selection criteria. The output of *qselect* is usually fed into other Sun Grid Engine commands to apply actions on a selected set of queues.

qsh(1)

qsh opens an interactive shell (in an *xterm(1)*) on a low loaded host. Any kind of interactive jobs can be run in this shell.

qstat(1)

qstat provides a status listing of all jobs and queues associated with the cluster.

qsub(1)

qsub is the user interface for submitting a job to Sun Grid Engine.

qtcsh(1)

qtcsh is a fully compatible replacement for the widely known and used Unix C-Shell (*csh*) derivative *tcsh*. It provides a command-shell with the extension of transparently distributing execution of designated applications to suitable and lightly loaded hosts via Sun Grid Engine.

SEE ALSO

sge_ckpt(1), *qacct(1)*, *qalter(1)*, *qconf(1)*, *qdel(1)*, *qhold(1)*, *qhost(1)*, *qlogin(1)*, *qmake(1)*, *qmod(1)*, *qmon(1)*, *qresub(1)*, *qrsls(1)*, *qrsh(1)*, *qselect(1)*, *qsh(1)*, *qstat(1)*, *qsub(1)*, *qtcsh(1)*, *Sun Grid Engine Installation and Administration Guide*, *Sun Grid Engine Quick Start Guide*, *Sun Grid Engine User's Guide*.

COPYRIGHT

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303-4900 U.S.A. All rights reserved.

This product or document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, AnswerBook2, docs.sun.com, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

NAME

Sun Grid Engine Checkpointing – the Sun Grid Engine checkpointing mechanism and checkpointing support

DESCRIPTION

Sun Grid Engine supports two levels of checkpointing: the user level and a operating system provided transparent level. User level checkpointing refers to applications, which do their own checkpointing by writing restart files at certain times or algorithmic steps and by properly processing these restart files when restarted.

Transparent checkpointing has to be provided by the operating system and is usually integrated in the operating system kernel. An example for a kernel integrated checkpointing facility is the CPR package for SGI IRIX platforms.

Checkpointing jobs need to be identified to the Sun Grid Engine system by using the *-ckpt* option of the *qsub(1)* command. The argument to this flag refers to a so called checkpointing environment, which defines the attributes of the checkpointing method to be used (see *checkpoint(5)* for details). Checkpointing environments are setup by the *qconf(1)* options *-ackpt*, *-dckpt*, *-mckpt* and *-sckpt*. The *qsub(1)* option *-c* can be used to overwrite the *when* attribute for the referenced checkpointing environment.

If a queue is of the type CHECKPOINTING, jobs need to have the checkpointing attribute flagged (see the **ckpt** option to *qsub(1)*) to be permitted to run in such a queue. As opposed to the behavior for regular batch jobs, checkpointing jobs are aborted under conditions, for which batch or interactive jobs are suspended or even stay unaffected. These conditions are:

- ❑ Explicit suspension of the queue or job via *qmod(1)* by the cluster administration or a queue owner if the *x* occasion specifier (see *qsub(1) -c* and *checkpoint(5)*) was assigned to the job.
- ❑ A load average value exceeding the migration threshold as configured for the corresponding queues (see *queue_conf(5)*).
- ❑ Shutdown of the Sun Grid Engine execution daemon *cod_execd(8)* being responsible for the checkpointing job.

After abortion, the jobs will migrate to other queues unless they were submitted to one specific queue by an explicit user request. The migration of jobs leads to a dynamic load balancing.

Note – The abortion of checkpointed jobs will free all resources (memory, swap space) which the job occupies at that time. This is opposed to the situation for suspended regular jobs, which still cover swap space.

RESTRICTIONS

When a job migrates to a queue on another machine at present no files are transferred automatically to that machine. This means that all files which are used throughout the entire job including restart files, executables and scratch files must be visible or transferred explicitly (e.g. at the beginning of the job script).

There are also some practical limitations regarding use of disk space for transparently checkpointing jobs. Checkpoints of a transparently checkpointed application are usually stored in a checkpoint file or directory by the operating system. The file or directory contains all the text, data, and stack space for the process, along with some additional control information. This means jobs which use a very large virtual address space will generate very large checkpoint files. Also the workstations on which the jobs will actually execute may have little free disk space. Thus it is not always possible to transfer a transparent checkpointing job to a machine, even though that machine is idle. Since large virtual memory jobs must wait for a machine that is both idle, and has a sufficient amount of free disk space, such jobs may suffer long turnaround times.

SEE ALSO

sge_intro(1), qconf(1), qmod(1), qsub(1), checkpoint(5), Sun Grid Engine Installation and Administration Guide, Sun Grid Engine User's Guide.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

qacct – report and account for Sun Grid Engine usage

SYNOPSIS

```
qacct [ -A Account ] [ -b BeginTime ] [ -d Days ]
      [ -e EndTime ] [ -g [GroupId|GroupName] ]
      [ -h [HostName] ] [ -help ] [ -history HistoryPath ]
      [ -j [JobId|JobName] ] [ -l attr=val,... ] [ -nohist ]
      [ -o [Owner] ] [ -pe [PEname] ] [ -q [Q_name] ]
      [ -slots [SlotNumber] ] [ -t task_id_range_list ]
      [ -P [Project] ] [ -D [Department] ] [ -f AcctFileName ]
```

DESCRIPTION

The *qacct* utility scans the accounting data file (see *accounting(5)*) and produces a summary of information for wall-clock time, cpu-time, and system time for the categories of hostname, queue-name, group-name, owner-name, job-name, job-ID and for the queues meeting the resource requirements as specified with the **-l** switch. Combinations of each category are permitted. Alternatively, all or specific jobs can be listed with the **-j** switch. For example the search criteria could include summarizing for a queue and an owner, but not for two queues in the same request.

OPTIONS

-A Account

The account for jobs to be summarized.

-b BeginTime

The earliest start time for jobs to be summarized, in the format [[CC]YY]MMDDhhmm[.SS]. See also **-d** option.

-d Days

The number of days to summarize and print accounting information on. If used together with the **-b BeginTime** option (see above), jobs started within **BeginTime** to **BeginTime + Days** are counted. If used together with the **-e EndTime** (see below) option, count starts at **EndTime - Days**.

-e EndTime

The latest start time for jobs to be summarized, in the format `[[CC]YY]MMDDhhmm[.SS]`. See also **-d** option.

[-f AcctFileName]

The accounting file to be used. If omitted, the system default accounting file is processed.

-g [GroupIdGroupName]

The numeric system group id or the group alphanumeric name of the job owners to be included in the accounting. If **GroupId/GroupName** is omitted, all groups are accounted.

-h [HostName]

The case-insensitive name of the host upon which accounting information is requested. If the name is omitted, totals for each host are listed separately.

-help

Display help information for the *qacct* command.

-history HistoryPath

The directory path where the *historical* queue and complexes configuration data is located, which is used for resource requirement matching in conjunction with the **-l** switch. If the latter is not set, this option is ignored.

-j [[JobNameJobId]]

The name or **ID** of the job during execution for which accounting information is printed. If neither a name nor an **ID** is given all jobs are enlisted.

This option changes the output format of *qacct*. If activated, CPU times are no longer accumulated but the “raw” accounting information is printed in a formatted form instead. See *accounting(5)* for an explanation of the displayed information.

-l attr=val,...

A resource requirement specification which must be met by the queues in which the jobs being accounted were executing. The matching is performed with *historical* data, i.e. it reflects the situation of the queue and complexes configuration at the time of the job start.

The resource request is very similar to the one described in *qsub(1)*. The main difference is that ever changing load information may not be requested as it is not contained in the historical configuration data being used.

-nohist

Only useful together with the **-l** option. It forces *qacct* not to use *historical* queue and complexes configuration data for resource requirement matching but instead retrieve actual queue and complexes configuration from *cod_qmaster(8)*.

Note – This may lead to confusing statistical results, as the current queue and complexes configuration may differ significantly from the situation being valid for past jobs.

Note – All hosts being referenced in the accounting file have to be up and running in order to get results.

–o [Owner]

The name of the owner of the jobs for which accounting statistics are assembled. If the optional **Owner** argument is omitted, a listing of the accounting statistics of all job owners being present in the accounting file is produced.

–pe [PEname]

The name of the parallel environment for which usage is summarized. If **PEname** is not given, accounting data is listed for each parallel environment separately.

–q [Q_name]

The name of the queue for which usage is summarized. If **Q_name** is not given, accounting data is listed for each queue separately.

–slots [SlotNumber]

The number of queue slots for which usage is summarized. If **SlotNumber** is not given, accounting data is listed for each number of queue slots separately.

–t task_id_range_list

Only available together with the **–j** option described above.

The **–t** switch specifies the job array task range, for which accounting information should be printed. Syntax and semantics of **task_id_range_list** are identical to that one described under the **–t** option to *qsub(1)*. Please see there also for further information on job arrays.

–P [Project]

The name of the project for which usage is summarized. If **Project** is not given, accounting data is listed for each owner project separately. Projects are only used when running in Sun Grid Engine, Enterprise Edition mode.

–D [Department]

The name of the department for which usage is summarized. If **Department** is not given, accounting data is listed for each owner department separately. Departments are only used when running in Sun Grid Engine, Enterprise Edition mode.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of `/usr/CODINE` is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qacct* uses (in the order of precedence):

- The name of the cell specified in the environment variable `COD_CELL`, if it is set.

- The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qacct* client resides. Per default the local host is used.

FILES

- `<codine_root>/<cell>/common/accounting`
Sun Grid Engine default accounting file
- `<codine_root>/<cell>/common/history`
Sun Grid Engine default history database
- `<codine_root>/<cell>/common/act_qmaster`
Sun Grid Engine master host file

SEE ALSO

sgc_intro(1), *qsub(1)*, *accounting(5)*, *cod_qmaster(8)*, *cod_commd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

qconf – Sun Grid Engine Queue Configuration

SYNTAX

qconf options

DESCRIPTION

Qconf allows the system administrator to add, delete, and modify the current Sun Grid Engine configuration, including queue management, host management, complex management and user management. *Qconf* also allows you to examine the current queue configuration for existing queues.

OPTIONS

Unless denoted otherwise, the following options and the corresponding operations are available to all users with a valid account.

-Attr obj_spec fname obj_instance,... *<add to object attributes>*

Similar to **-attr** (see below) but takes specifications for the object attributes to be enhanced from file named **fname**. As opposed to **-attr**, multiple attributes can be enhanced. Their specification has to be enlisted in **fname** following the file format of the corresponding object (see *queue_conf(5)* for the queue, for example).

Requires root/manager privileges.

-Ac complex_name fname *<add complex>*

Add the complex **complex_name** defined in **fname** to the Sun Grid Engine cluster. The format of the complex specification is described in *complex(5)*. Requires root or manager privileges.

-Acal fname *<add calendar>*

Adds a new calendar definition to the Sun Grid Engine environment. Calendars are used in Sun Grid Engine for defining availability and unavailability schedules of queues. The format of a calendar definition is described in *calendar_conf(5)*.

The calendar definition is taken from the file **fname**. Requires root/ manager privileges.

- Ackpt *fname*** *<add ckpt. environment>*
- Add the checkpointing environment as defined in **fname** (see *checkpoint(5)*) to the list of supported checkpointing environments. Requires root or manager privileges.
- Aconf *file_list*** *<add configurations>*
- Add the cluster configurations (see *codine_conf(5)*) specified in the files enlisted in the comma separated **file_list**.
- Requires root or manager privileges.
- Ae *fname*** *<add execution host>*
- Add the execution host defined in **fname** to the Sun Grid Engine cluster. The format of the execution host specification is described in *host_conf(5)*. Requires root or manager privileges.
- Ap *fname*** *<add PE configuration>*
- Add the parallel environment (PE) defined in **fname** to the Sun Grid Engine cluster. Requires root or manager privileges.
- Aprj *fname*** *<add new project>*
- This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
- Adds the project description defined in **fname** to the list of registered projects (see *project(5)*). Requires root or manager privileges.
- Aq *fname*** *<add new queue>*
- Add the queue defined in **fname** to the Sun Grid Engine cluster. Requires root or manager privileges.
- Au *fname*** *<add ACL>*
- Adds a user access list (ACL) to Sun Grid Engine. User lists are used for queue usage authentication. Requires root/manager/operator privileges.
- Dattr *obj_spec fname obj_instance,...*** *<del. from object attribs>*
- Similar to **-dattr** (see below) but the definition of the list attributes from which entries are to be deleted is contained in the file named **fname**. As opposed to **-dattr**, multiple attributes can be modified. Their specification has to be enlisted in **fname** following the file format of the corresponding object (see *queue_conf(5)* for the queue, for example).
- Requires root/manager privileges.
- Mattr *obj_spec fname obj_instance,...*** *<mod. object attributes>*
- Similar to **-mattr** (see below) but takes specifications for the object attributes to be modified from file named **fname**. As opposed to **-mattr**, multiple attributes can be modified. Their specification has to be enlisted in **fname** following the file format of the corresponding object (see *queue_conf(5)* for the queue, for example).
- Requires root/manager privileges.

-Mc complex_name fname

<modify complex>

Overwrites the specified complex by the contents of **fname**. The argument file must comply to the format specified in *complex(5)*. Requires root or manager privilege.

-Mcal fname

<modify calendar>

Overwrites the calendar definition as specified in **fname**. The argument file must comply to the format described in *calendar_conf(5)*. Requires root or manager privilege.

-Mckpt fname

<modify ckpt. environment>

Overwrite an existing checkpointing environment with the definitions in **fname** (see *checkpoint(5)*). The name attribute in **fname** has to match an existing checkpointing environment. Requires root or manager privileges.

-Me fname

<modify execution host>

Overwrites the execution host configuration for the specified host with the contents of **fname**, which must comply to the format defines in *host_conf(5)*. Requires root or manager privilege.

-Mp fname

<modify PE configuration>

Same as **-mp** (see below) but instead of invoking an editor to modify the PE configuration the file **fname** is considered to contain a changed configuration. Refer to *sge_pe(5)* for details on the PE configuration format. Requires root or manager privilege.

-Mprj fname

<modify project config.>

Same as **-mprj** (see below) but instead of invoking an editor to modify the project configuration the file **fname** is considered to contain a changed configuration. Refer to *project(5)* for details on the project configuration format. Requires root or manager privilege.

-Mq fname

<modify queue configuration>

Same as **-mq** (see below) but instead of invoking an editor to modify the queue configuration the file **fname** is considered to contain a changed configuration. Refer to *queue_conf(5)* for details on the queue configuration format. Requires root or manager privilege.

-Mqattr fname q_name,...

<modify queue attributes>

DEPRECATED: Use **-Mattr!**

Allows changing of selected queue configuration attributes in multiple queues with a single command. In all queues contained in the comma separated queue name list the queue attribute definitions contained in **fname** will be applied. Queue attributes not contained in **fname** will be left unchanged.

All queue attributes can be modified except for *qname* and *qhostname*. Refer to *queue_conf(5)* for details on the queue configuration format. Requires root or manager privilege.

-Mu fname

<modify ACL>

Takes the user access list (ACL) defined in **fname** to overwrite any existing ACL with the same name. See *access_list(5)* for information on the ACL configuration format. Requires root or manager privilege.

-Rattr obj_spec fname obj_instance,...

<replace object attribs>

Similar to **-rattr** (see below) but the definition of the list attributes whose content is to be replaced is contained in the file named **fname**. As opposed to **-rattr**, multiple attributes can be modified. Their specification has to be enlisted in **fname** following the file format of the corresponding object (see *queue_conf(5)* for the queue, for example).

Requires root/manager privileges.

-aattr obj_spec attr_name val obj_instance,...

<add to object attributes>

Allows adding specifications to a single configuration list attribute in multiple instances of an object with a single command. Currently supported objects are the queue and the host configuration being specified as *queue* or *host* in **obj_spec**. The queue **load_thresholds** parameter is an example of a list attribute. With the **-aattr** option, entries can be added to such lists, while they can be deleted with **-dattr**, modified with **-mattr**, and replaced with **-rattr**.

The name of the configuration attribute to be enhanced is specified with **attr_name** followed by **val** as a *name=value* pair. The comma separated list of object instances (e.g., the list of queues) to which the changes have to be applied are specified at the end of the command.

The following restriction applies: For the *host* object the **load_values** attribute cannot be modified (see *host_conf(5)*).

Requires root or manager privilege.

-ac complex_name

<add complex>

Adds a complex to the Sun Grid Engine environment. Complex entries contain one or more resources which may be requested by jobs submitted to the system. The *complex(5)* manual page contains detailed information about the format of a complex definition.

When using the **-ac** option the complex name is given in the command option. *Qconf* will then open a temporary file and start up the text editor indicated by the environment variable **EDITOR** (default editor is *vi(1)* if **EDITOR** is not set). After entering the complex definition and closing the editor the new complex is checked and registered with *cod_qmaster(8)*. Requires root/manager privileges.

-acal calendar_name

<add calendar>

Adds a new calendar definition to the Sun Grid Engine environment. Calendars are used in Sun Grid Engine for defining availability and unavailability schedules of queues. The format of a calendar definition is described in *calendar_conf(5)*.

With the calendar name given in the option argument *qconf* will open a temporary file and start up the text editor indicated by the environment variable **EDITOR** (default editor is *vi(1)* if **EDITOR** is not set). After entering the calendar definition and closing the editor the new calendar is checked and registered with *cod_qmaster(8)*. Requires root/manager privileges.

-ackpt ckpt_name

<add ckpt. environment>

Adds a checkpointing environment under the name **ckpt_name** to the list of checkpointing environments maintained by Sun Grid Engine and to be usable to submit checkpointing jobs (see *checkpoint(5)* for details on the format of a checkpointing environment definition). *Qconf* retrieves a default checkpointing

environment configuration and executes *vi(1)* (or **\$EDITOR** if the **EDITOR** environment variable is set) to allow you to customize the checkpointing environment configuration. Upon exit from the editor, the checkpointing environment is registered with *cod_qmaster(8)*. Requires root/manager privileges.

-aconf host,... <add configuration>

Successively adds cluster configurations (see *sge_conf(5)*) For the hosts in the comma separated *file_list*. For each host, an editor (**\$EDITOR** indicated or *vi(1)*) is invoked and the configuration for the host can be entered. The configuration is registered with *cod_qmaster(8)* after saving the file and quitting the editor.

Requires root or manager privileges.

-ae [host_template] <add execution host>

Adds a host to the list of Sun Grid Engine execution hosts. If a queue is configured on a host this host is automatically added to the Sun Grid Engine execution host list. Adding execution hosts explicitly offers the advantage to be able to specify parameters like load scale values with the registration of the execution host. However, these parameters can be modified (from their defaults) at any later time via the **-me** option described below.

If the **host_template** argument is present, *qconf* retrieves the configuration of the specified execution host from *cod_qmaster(8)* or a generic template otherwise. The template is then stored in a file and *qconf* executes *vi(1)* (or the editor indicated by **\$EDITOR** if the **EDITOR** environment variable is set) to change the entries in the file. The format of the execution host specification is described in *host_conf(5)*. When the changes are saved in the editor and the editor is quit the new execution host is registered with *cod_qmaster(8)*. Requires root/manager privileges.

-ah hostname,... <add administrative host>

Adds hosts **hostname** to the Sun Grid Engine trusted host list (a host must be in this list to execute administrative Sun Grid Engine commands, the sole exception to this being the execution of *qconf* on the *cod_qmaster(8)* node). The default Sun Grid Engine installation procedures usually add all designated execution hosts (see the **-ae** option above) to the Sun Grid Engine trusted host list automatically. Requires root or manager privileges.

-am user,... <add managers>

Adds the indicated users to the Sun Grid Engine manager list. Requires root or manager privileges.

-ao user,... <add operators>

Adds the indicated users to the Sun Grid Engine operator list. Requires root/manager privileges.

-ap pe_name <add new PE>

Adds a *Parallel Environment* (PE) description under the name **pe_name** to the list of PEs maintained by Sun Grid Engine and to be usable to submit parallel jobs (see *sge_pe(5)* for details on the format of a PE definition). *Qconf* retrieves a default PE configuration and executes *vi(1)* (or **\$EDITOR** if the **EDITOR** environment variable is set) to allow you to customize the PE configuration. Upon exit from the editor, the PE is registered with *cod_qmaster(8)*. Requires root/manager privileges.

-apri

<add new project>

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Adds a project description to the list of registered projects (see *project(5)*). *Qconf* retrieves a template project configuration and executes *vi(1)* (or *\$EDITOR* if the *EDITOR* environment variable is set) to allow you to customize the new project. Upon exit from the editor, the template is registered with *cod_qmaster(8)*. Requires root or manager privileges.

-aq [q_template]

<add new queue>

Qconf retrieves either the default queue configuration (see *queue_conf(5)*) or the configuration of the queue **q_template** (if the optional argument is present) and executes *vi(1)* (or *\$EDITOR* if the *EDITOR* environment variable is set) to allow you to customize the queue configuration. Upon exit from the editor, the queue is registered with *cod_qmaster(8)*. A minimal configuration requires only that the queue name and queue hostname be set. Requires root or manager privileges.

-as hostname,...

<add submit hosts>

Add hosts **hostname** to the list of hosts allowed to submit Sun Grid Engine jobs and control their behavior only. Requires root or manager privileges.

-astnode node_path=shares,...

<add share tree node>

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Adds the specified share tree node(s) to the share tree (see *share_tree(5)*). The **node_path** is a hierarchical path (**[/]*node_name*[/*.node_name*...]**) specifying the location of the new node in the share tree. The base name of the *node_path* is the name of the new node. The node is initialized to the number of specified shares. Requires root or manager privileges.

-astree

<add share tree>

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Adds the definition of a share tree to the system (see *share_tree(5)*). A template share tree is retrieved and an editor (either *vi(1)* or the editor indicated by *\$EDITOR*) is invoked for modifying the share tree definition. Upon exiting the editor, the modified data is registered with *cod_qmaster(8)*. Requires root or manager privileges.

-au user,... acl_name,...

<add users to ACLs>

Adds users to Sun Grid Engine user access lists (ACLs). User lists are used for queue usage authentication. Requires root/manager/operator privileges.

-auser *<add user>*

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Adds a user to the list of registered users (see *user(5)*). This command invokes an editor (either *vi(1)* or the editor indicated by the **EDITOR** environment variable) for a template user. The new user is registered after changing the entry and exiting the editor. Requires root or manager privileges.

-cq queue_name,... *<clean queue>*

Cleans queue from jobs which haven't been reaped. Primarily a development tool. Requires root/manager/operator privileges.

-dattr obj_spec attr_name val obj_instance,... *<delete in object attributes>*

Allows deleting specifications in a single configuration list attribute in multiple instances of an object with a single command. Currently supported objects are the queue and the host configuration being specified as *queue* or *host* in **obj_spec**. The queue **load_thresholds** parameter is an example of a list attribute. With the **-dattr** option, entries can be deleted from such lists, while they can be added with **-aattr**, modified with **-mattr**, and replaced with **-rattr**.

The name of the configuration attribute to be modified is specified with **attr_name** followed by **val** defining the name of the attribute list entry to be deleted. The comma separated list of object instances (e.g., the list of queues) to which the changes have to be applied are specified at the end of the command.

The following restriction applies: For the *host* object the **load_values** attribute cannot be modified (see *host_conf(5)*).

Requires root or manager privilege.

-dc complex_name,... *<delete complex>*

Deletes complexes from Sun Grid Engine. Requires root/manager privileges.

-dcal calendar_name,... *<delete calendar>*

Deletes the specified calendar definition from Sun Grid Engine. Requires root/manager privileges.

-dckpt ckpt_name *<delete ckpt. environment>*

Deletes the specified checkpointing environment. Requires root/manager privileges.

-dconf host,... *<delete configuration>*

The configuration entry for the specified hosts is deleted from the configuration list. Requires root or manager privilege.

-de host_name,... *<delete execution host>*

Deletes hosts from the Sun Grid Engine execution host list. Requires root/manager privileges.

-dh host_name,... *<delete administrative host>*

Deletes hosts from the Sun Grid Engine trusted host list. The host on which *cod_qmaster(8)* is currently running cannot be removed from the list of administrative hosts. Requires root/manager privileges.

- dm user[,user,...]** *<delete managers>*
 Deletes managers from the manager list. Requires root/manager privileges.
- do user[,user,...]** *<delete operators>*
 Deletes operators from the operator list. Requires root/manager privileges.
- dp pe_name** *<delete parallel environment>*
 Deletes the specified parallel environment (PE). Requires root/manager privileges.
- dprj project,...** *<delete projects>*
 This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
 Deletes the specified project(s). Requires root/manager privileges.
- dq queue_name,...** *<delete queue>*
 Removes the specified queue(s). Active jobs will be allowed to run to completion. Requires root/manager privileges.
- ds host_name,...** *<delete submit host>*
 Deletes hosts from the Sun Grid Engine submit host list. Requires root/manager privileges.
- dstnode node_path,...** *<delete share tree node>*
 This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
 Deletes the specified share tree node(s). The **node_path** is a hierarchical path ([/]**node_name**[/**node_name**...]) specifying the location of the node to be deleted in the share tree. Requires root or manager privileges.
- dstree** *<delete share tree>*
 This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
 Deletes the current share tree. Requires root or manager privileges.
- du user,... acl_name,...** *<delete users from ACL>*
 Deletes one or more users from one or more Sun Grid Engine user access lists (ACLs). Requires root/manager/operator privileges.
- dul acl_name,...** *<delete user lists>*
 Deletes one or more user lists from the system. Requires root/manager/operator privileges.
- duser user,...** *<delete users>*
 This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
 Deletes the specified user(s) from the list of registered users. Requires root or manager privileges.

-help

Prints a listing of all options.

-k{m|s|e[j]} [host,...]

<shutdown Sun Grid Engine>

Used to shutdown Sun Grid Engine components (daemons). In the form **-km** *cod_qmaster(8)* is forced to terminate in a controlled fashion. In the same way the **-ks** switch causes termination of *cod_schedd(8)*. Shutdown of all running *cod_execd(8)* processes currently registered is initiated by the **-ke** option. If **-kej** is specified instead, all jobs running on the execution hosts are aborted prior to termination of the corresponding *cod_execd(8)*. The optional comma separated host list specifies the execution hosts to be addressed by the **-ke** and **-kej** option.

Requires root or manager privileges.

-mattr obj_spec attr_name val obj_instance,...

<modify object attributes>

Allows changing a single configuration attribute in multiple instances of an object with a single command. Currently supported objects are the queue and the host configuration being specified as *queue* or *host* in **obj_spec**.

Note – "-mattr queue attr_name val q_name, ..." is equivalent to "-mqattr attr_name val q_name,..." (see below). The latter is available for backward compatibility.

The name of the configuration attribute to be modified is specified with **attr_name** followed by the value to which the attribute is going to be set. If the attribute is a list, such as the queue **load_thresholds**, **val** can be a *name=value* pair, in which case only a corresponding entry in the list is changed. Refer to the **-aattr**, **-dattr** and **-rattr** options for a description of further means to change specifically such list attributes.

The comma separated list of object instances (e.g., the list of queues) to which the changes have to be applied are specified at the end of the command.

The following restrictions apply: For the *queue* object the **qname** and **qhostname** attributes cannot be modified (see *queue_conf(5)*). For the *host* object the **hostname**, **load_values** and **processors** attributes cannot be modified (see *host_conf(5)*).

Requires root or manager privilege.

-mc complex_name

<modify complex>

The specified complex configuration (see *complex(5)*) is retrieved, an editor is executed (either *vi(1)* or the editor indicated by \$EDITOR) and the changed complex configuration is registered with *cod_qmaster(8)* upon exit of the editor. Requires root or manager privilege.

-mcal calendar_name

<modify calendar>

The specified calendar definition (see *calendar_conf(5)*) is retrieved, an editor is executed (either *vi(1)* or the editor indicated by \$EDITOR) and the changed calendar definition is registered with *cod_qmaster(8)* upon exit of the editor. Requires root or manager privilege.

-mckpt ckpt_name <modify ckpt. environment>

Retrieves the current configuration for the specified checkpointing environment, executes an editor (either *vi(1)* or the editor indicated by the **EDITOR** environment variable) and registers the new configuration with the *cod_qmaster(8)*. Refer to *checkpoint(5)* for details on the checkpointing environment configuration format. Requires root or manager privilege.

-mconf [host,...lglobal] <modify configuration>

The configuration for the specified host is retrieved, an editor is executed (either *vi(1)* or the editor indicated by **\$EDITOR**) and the changed configuration is registered with *cod_qmaster(8)* upon exit of the editor. If the optional host argument is omitted or if the special host name “global” is specified, the cell global configuration is modified. The format of the host configuration is described in *codine_conf(5)*. Requires root or manager privilege.

-me hostname <modify execution host>

Retrieves the current configuration for the specified execution host, executes an editor (either *vi(1)* or the editor indicated by the **EDITOR** environment variable) and registers the changed configuration with *cod_qmaster(8)* upon exit from the editor. The format of the execution host configuration is described in *host_conf(5)*. Requires root or manager privilege.

-mp pe_name <modify PE configuration>

Retrieves the current configuration for the specified *parallel environment* (PE), executes an editor (either *vi(1)* or the editor indicated by the **EDITOR** environment variable) and registers the new configuration with the *cod_qmaster(8)*. Refer to *sge_pe(5)* for details on the PE configuration format. Requires root or manager privilege.

-mprj project <modify project>

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Data for the specific project is retrieved (see *project(5)*) and an editor (either *vi(1)* or the editor indicated by **\$EDITOR**) is invoked for modifying the project definition. Upon exiting the editor, the modified data is registered. Requires root or manager privileges.

-mq queue_name <modify queue configuration>

Retrieves the current configuration for the specified queue, executes an editor (either *vi(1)* or the editor indicated by the **EDITOR** environment variable) and registers the new configuration with the *cod_qmaster(8)*. Refer to *queue_conf(5)* for details on the queue configuration format. Requires root or manager privilege.

-mqattr attr_name val q_name,... <modify queue attributes>

DEPRECATED: Use **-mattr!**

Allows changing of a single queue configuration attribute in multiple queues with a single command. In all queues contained in the comma separated queue name list the value of the attribute **attr_name** will be overwritten with **val**.

All queue attributes can be modified except for *qname* and *ghostname*. Refer to *queue_conf(5)* for details on the queue configuration format. Requires root or manager privilege.

-msconf *<modify scheduler configuration>*

The current scheduler configuration (see *sched_conf(5)*) is retrieved, an editor is executed (either *vi(1)* or the editor indicated by *\$EDITOR*) and the changed configuration is registered with *cod_qmaster(8)* upon exit of the editor. Requires root or manager privilege.

-mstnode node_path=shares,... *<modify share tree node>*

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Modifies the specified share tree node(s) in the share tree (see *share_tree(5)*). The **node_path** is a hierarchical path (*[/node_name[/node_name...]]*) specifying the location of an existing node in the share tree. The node is set to the number of specified **shares**. Requires root or manager privileges.

-mstree *<modify share tree>*

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Modifies the definition of the share tree (see *share_tree(5)*). The present share tree is retrieved and an editor (either *vi(1)* or the editor indicated by *\$EDITOR*) is invoked for modifying the share tree definition. Upon exiting the editor, the modified data is registered with *cod_qmaster(8)*. Requires root or manager privileges.

-mu acl_name *<modify user access lists>*

Retrieves the current configuration for the specified user access list, executes an editor (either *vi(1)* or the editor indicated by the *EDITOR* environment variable) and registers the new configuration with the *cod_qmaster(8)*. Requires root or manager privilege.

-muser user *<modify user>*

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Data for the specific user is retrieved (see *user(5)*) and an editor (either *vi(1)* or the editor indicated by the *EDITOR* environment variable) is invoked for modifying the user definition. Upon exiting the editor, the modified data is registered. Requires root or manager privileges.

-rattr obj_spec attr_name val obj_instance,... *<replace object attributes>*

Allows replacing a single configuration list attribute in multiple instances of an object with a single command. Currently supported objects are the queue and the host configuration being specified as *queue* or *host* in **obj_spec**. The queue **load_thesholds** parameter is an example of a list attribute. With the **-rattr** option, such lists can be replaced, while entries can be added to them with **-aattr**, deleted with **-dattr**, and modified with **-mattr**.

The name of the configuration attribute to be modified is specified with **attr_name** followed by **val** defining the new setting of the attribute. The comma separated list of object instances (e.g., the list of

queues) to which the changes have to be applied are specified at the end of the command.

The following restriction applies: For the *host* object the **load_values** attribute cannot be modified (see *host_conf(5)*).

Requires root or manager privilege.

- | | |
|---|--|
| -sc complex_name,... | <i><show complexes></i> |
| Display the configuration of one or more complexes. | |
| -scal calendar_name | <i><show calendar></i> |
| Display the configuration of the specified calendar. | |
| -scall | <i><show calendar list></i> |
| Show a list of all calendars currently defined. | |
| -scl | <i><show complex list names></i> |
| Show a list of all complexes currently configured. | |
| -sckpt ckpt_name | <i><show ckpt. environment></i> |
| Display the configuration of the specified checkpointing environment. | |
| -sckptl | <i><show ckpt. environment list></i> |
| Show a list of the names of all checkpointing environments currently configured. | |
| -sconf [host,... global] | <i><show configuration></i> |
| Print the cluster configuration being in effect globally or on specified host(s). If the optional comma separated host list argument is omitted or the special string global is given, the global cell configuration is displayed. For any other hostname in the list the merger of the global configuration and the host specific configuration is displayed. The format of the host configuration is described in <i>sge_conf(5)</i> . | |
| -sconfl | <i><show configuration list></i> |
| Display a list of hosts for which configurations are available. The special host name “global” refers to the cell global configuration. | |
| -se hostname | <i><show execution host></i> |
| Displays the definition of the specified execution host. | |
| -sel | <i><show execution hosts></i> |
| Displays the Sun Grid Engine execution host list. | |
| -sep | <i><show licensed processors></i> |
| Displays a list of number of processors which are licensed per execution host and in total. | |
| -sh | <i><show administrative hosts></i> |
| Displays the Sun Grid Engine administrative host list. | |

- sm** <show managers>
Displays the managers list.
- so** <show operators>
Displays the operator list.
- sp pe_name** <show PE configuration>
Show the definition of the *parallel environment* (PE) specified by the argument.
- spl** <show PE-list>
Show a list of all currently defined *parallel environments* (PEs).
- sprj project** <show project>
This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
Shows the definition of the specified project (see *project(5)*).
- sprjl** <show project list>
This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
Shows the list of all currently defined projects.
- sq queue_name[,queue_name,...]** <show queues>
Displays one or multiple queues.
- sql** <show queue list>
Show a list of all currently defined queues.
- ss** <show submit hosts>
Displays the Sun Grid Engine submit host list.
- ssconf** <show scheduler configuration>
Displays the current scheduler configuration in the format explained in *sched_conf(5)*.
- sstnode node_path,...** <show share tree node>
This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
Shows the name and shares of the specified share tree node(s) (see *share_tree(5)*). The **node_path** is a hierarchical path ([/]**node_name**[/.]**node_name**...) specifying the location of a node in the share tree.
- sstree** <show share tree>
This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
Shows the definition of the share tree (see *share_tree(5)*).

- sss** *<show scheduler status>*
- Currently displays the host on which the Sun Grid Engine scheduler is active or an error message if no scheduler is running.
- su acl_name** *<show user ACL>*
- Displays a Sun Grid Engine user access list (ACL).
- sul** *<show user lists>*
- Displays a list of names of all currently defined Sun Grid Engine user access lists (ACLs).
- suser user,...** *<show user>*
- This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
- Shows the definition of the specified user(s) (see *user(5)*).
- suserl** *<show users>*
- This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.
- Shows the list of all currently defined users.
- tsm** *<trigger scheduler monitoring>*
- The Sun Grid Engine scheduler *cod_schedd(8)* is forced by this option to print trace messages of its next scheduling run to the file *<codine_root>/<cell>/common/schedd_runlog*. The messages indicate the reasons for jobs and queues not being selected in that run. Requires root or manager privileges.

Note – The reasons for job requirements being invalid with respect to resource availability of queues are displayed using the format as described for the *qstat(1)* -F option (see description of Full Format in section OUTPUT FORMATS of the *qstat(1)* manual page).

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of /usr/CODINE is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qconf* uses (in the order of precedence):

The name of the cell specified in the environment

variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qconf* client resides. Per default the local host is used.

RESTRICTIONS

Modifications to a queue configuration do not affect an active queue, taking effect on next invocation of the queue (i.e., the next job).

FILES

<codine_root>/<cell>/common/act_qmaster
Sun Grid Engine master host file

SEE ALSO

sgc_intro(1), *qstat(1)*, *checkpoint(5)*, *complex(5)*, *sgc_conf(5)*, *host_conf(5)*, *sgc_pe(5)*, *queue_conf(5)*, *cod_execd(8)*, *cod_qmaster(8)*, *cod_schedd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

`qdel` – delete Sun Grid Engine jobs from queues

SYNTAX

`qdel [-f] [-help] [-verify] [job/task_id_list]`

`qdel [-f] [-help] [-verify] -u user_list | -uall`

DESCRIPTION

Qdel provides a means for a user/operator/manager to delete one or more jobs. *Qdel* deletes jobs in the order in which their job identifiers are presented.

OPTIONS

-f

Force action for running jobs. The job(s) are deleted from the list of jobs registered at *cod_qmaster*(8) even if the *cod_execd*(8) controlling the job(s) does not respond to the delete request sent by *cod_qmaster*(8).

Users which are neither Sun Grid Engine managers nor operators can only use the **-f** option (for their own jobs) if the cluster configuration entry **qmaster_params** contains the flag **ENABLE_FORCED_QDEL** (see *sge_conf*(5)). However, behavior for administrative and non-administrative users differs. Jobs are deleted from the Sun Grid Engine database immediately in case of administrators. Otherwise, a regular deletion is attempted first and a forced cancellation is only executed if the regular deletion was unsuccessful.

-help

Prints a listing of all options.

-u username,... | -uall

Deletes only those jobs which were submitted by users specified in the list of usernames. For managers it is possible to use the **qdel -uall** command to delete all jobs of all users.

If you use the **-u** or **-uall** switch it is permitted to specify a additional *job/task_id_list*.

-verify

performs no modifications but just prints what would be done if **-verify was not present**.

job/task_id_list

Specified by the following form:

job_id[.task_range][,job_id[.task_range],...]

If present, the *task_range* restricts the effect of the *qdel* operation to the job array task range specified as suffix to the job id (see the **-t** option to *qsub(1)* for further details on job arrays).

The task range specifier has the form *n[-m[:s]]*. The range may be a single number, a simple range of the form *n-m* or a range with a step size.

Instead of *job/task_id_list* it is possible to use the keyword 'all' to modify all jobs of the current user.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qdel* uses (in the order of precedence):

- The name of the cell specified in the environment variable **COD_CELL**, if it is set.

- The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qdel* client resides. Per default the local host is used.

FILES

<cod_root>/<cell>/common/act_qmaster
Sun Grid Engine master host file

SEE ALSO

sge_intro(1), *qstat(1)*, *qsub(1)*, *cod_qmaster(8)*, *cod_execd(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

qhold – hold back Sun Grid Engine jobs from execution

SYNTAX

qhold [**-h** {ulols},...] [**-help**] [**job/task_id_list**]

qhold [**-h** {ulols},...] [**-help**] **-u** **user_list** | **-uall**

DESCRIPTION

Qhold provides a means for a user/operator/manager to place so called *holds* on one or more jobs pending to be scheduled for execution. As long as any type of hold is assigned to a job, the job is not eligible for scheduling.

Holds can be removed with the *qrls(1)* or the *qalter(1)* command.

There are three different types of holds:

user

User holds can be assigned and removed by managers, operators and the owner of the jobs.

operator

Operator holds can be assigned and removed by managers and operators.

system

System holds can be assigned and removed by managers only.

If no hold type is specified with the **-h** option (see below) the user hold is assumed by default.

An alternate way to assign holds to jobs is the *qsub(1)* or the *qalter(1)* command (see the **-h** option).

OPTIONS

-h {ulols},...

Assign a u(ser), o(perator) or s(system) hold or a combination thereof to one or more jobs.

-help

Prints a listing of all options.

-u username,... | -uall

Changes are only made on those jobs which were submitted by users specified in the list of usernames. For managers it is possible to use the **qhold -uall** command to set a hold for all jobs of all users.

If you use the **-u** or **-uall** switch it is permitted to specify a additional *job/task_id_list*.

job/task_id_list

Specified by the following form:

job_id[.task_range][,job_id[.task_range],...]

If present, the *task_range* restricts the effect of the *qhold* operation to the job array task range specified as suffix to the job id (see the **-t** option to *qsub(1)* for further details on job arrays).

The task range specifier has the form *n[-m[:s]]*. The range may be a single number, a simple range of the form *n-m* or a range with a step size.

Instead of *job/task_id_list* it is possible to use the keyword 'all' to modify the hold state for all jobs of the current user.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qhold* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qhold* client resides. Per default the local host is used.

SEE ALSO

sgc_intro(1), *qalter(1)*, *qrls(1)*, *qsub(1)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

qhost – show the status of Sun Grid Engine hosts, queues, jobs

SYNTAX

```
qhost [ -F [resource_name,...] [ -help ] [ -h host_list ]  
        [ -j ] [ -l resource=val,... ] [ -u user,... ]
```

DESCRIPTION

qhost shows the current status of the available Sun Grid Engine hosts, queues and the jobs associated with the queues. Selection options allow you to get information about specific hosts, queues, jobs or users. Without any option *qhost* will display a list of all hosts without queue or job information.

OPTIONS

-F [**resource_name**,...]

qhost will present a detailed listing of the current resource availability per host with respect to all resources (if the option argument is omitted) or with respect to those resources contained in the *resource_name* list. Please refer to the description of the **Full Format** in section **OUTPUT FORMATS** below for further detail.

-help

Prints a listing of all options.

-h **host_list**

Prints a list of all hosts contained in *host_list*.

-j

Prints all jobs running on the queues hosted by the shown hosts. This switch calls **-q** implicitly.

-l **resource[=value]**,...

Defines the resources required by the hosts on which information is requested. Matching is performed on hosts.

-q

Show information about the queues hosted by the displayed hosts.

-u user,...

Display information only on those jobs and queues being associated with the users from the given user list.

OUTPUT FORMATS

Depending on the presence or absence of the **-q** or **-F** and **-j** option three output formats need to be differentiated. PP

Default Format (without **-q**, **-F** and **-j**)

Following the header line a line is printed for each host consisting of

- the Hostname
- the Architecture.
- the Number of processors.
- the Load.
- the Total Memory.
- the Used Memory.
- the Total Swapspace.
- the Used Swapspace.

If the **-q** option is supplied, each host status line also contains extra lines for every queue hosted by the host consisting of,

- the queue name.
- the queue type – one of B(atch), I(nteractive), C(heckpointing), P(arallel), T(ransfer) or combinations thereof,
- the number of used and available job slots,
- the state of the queue – one of u(nknown) if the corresponding *cod_execd(8)* cannot be contacted, a(larm), A(larm), C(alendar suspended), s(uspended), S(ubordinate), d(isabled), D(isabled), E(rror) or combinations thereof.

If the state is a(alarm) at least one of the load thresholds defined in the *load_thresholds* list of the queue configuration (see *queue_conf(5)*) is currently exceeded, which prevents from scheduling further jobs to that queue.

As opposed to this, the state A(larm) indicates that at least one of the suspend thresholds of the queue (see *queue_conf(5)*) is currently exceeded. This will result in jobs running in that queue being successively suspended until no threshold is violated.

The states s(uspended) and d(isabled) can be assigned to queues and released via the *qmod(1)* command. Suspending a queue will cause all jobs executing in that queue to be suspended.

The states D(isabled) and C(alendar suspended) indicate that the queue has been disabled or suspended automatically via the calendar facility of Sun Grid Engine (see *calendar_conf(5)*), while the S(ubordinate) state indicates, that the queue has been suspend via subordination to another queue (see *queue_conf(5)* for details). When suspending a queue (regardless of the cause) all jobs executing in that queue are suspended too.

If an E(rror) state is displayed for a queue, *cod_execd(8)* on that host was unable to locate the *cod_shepherd(8)* executable on that host in order to start a job. Please check the error logfile of that *cod_execd(8)* for leads on how to resolve the problem. Please enable the queue afterwards via the **-c** option of the *qmod(1)* command manually.

If the **-F** option was used, resource availability information is printed following the host status line. For each resource (as selected in an option argument to **-F** or for all resources if the option argument was omitted) a single line is displayed with the following format:

- a one letter specifier indicating whether the current resource availability value was dominated by either
 - **'g'** - a cluster global,
 - **'h'** - a host total or
- a second one letter specifier indicating the source for the current resource availability value, being one of
 - **'l'** - a load value reported for the resource,
 - **'L'** - a load value for the resource after administrator defined load scaling has been applied,
 - **'c'** - availability derived from the consumable resources facility (see *complexes(5)*), **'v'** - a default complexes configuration value never overwritten by a load report or a consumable update or
 - **'f'** - a fixed availability definition derived from a non-consumable complex attribute or a fixed resource limit.
- after a colon the name of the resource on which information is displayed.
- after an equal sign the current resource availability value.

The displayed availability values and the sources from which they derive are always the minimum values of all possible combinations. Hence, for example, a line of the form "qf:h_vmem=4G" indicates that a queue currently has a maximum availability in virtual memory of 4 Gigabyte, where this value is a fixed value (e.g. a resource limit in the queue configuration) and it is queue dominated, i.e. the host in total may have more virtual memory available than this, but the queue doesn't allow for more. Contrarily a line "hl:h_vmem=4G" would also indicate an upper bound of 4 Gigabyte virtual memory availability, but the limit would be derived from a load value currently reported for the host. So while the queue might allow for jobs with higher virtual memory requirements, the host on which this particular queue resides currently only has 4 Gigabyte available.

After the queue status line (in case of **-j**) a single line is printed for each job running currently in this queue. Each job status line contains

- the job ID,
- the job name,
- the job owner name,
- the status of the job – one of t(ransferring), r(unning), R(estarted), s(uspended), S(uspended) or T(hreshold) (see the **Reduced Format** section for detailed information),
- the start date and time and the function of the job (MASTER or SLAVE - only meaningful in case of a parallel job) and
- the priority of the jobs.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of /usr/CODINE is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qstat* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qstat* client resides. Per default the local host is used.

FILES

<codine_root>/<cell>/common/act_qmaster
Sun Grid Engine master host file

SEE ALSO

sgc_intro(1), qalter(1), qconf(1), qhold(1), qmod(1), qstat(1), qsub(1), queue_conf(5), cod_commd(8), cod_execd(8), cod_qmaster(8), cod_qstd(8), cod_shepherd(8).

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

qmake – distributed parallel make, scheduling by Sun Grid Engine.

SYNTAX

qmake [options] -- [gmake options]

DESCRIPTION

Qmake is a parallel, distributed *make(1)* utility. Scheduling of the parallel *make* tasks is done by Sun Grid Engine. It is based on *gmake* (GNU make), version 3.78.1. Both Sun Grid Engine and *gmake* commandline options can be specified. They are separated by "--".

All Sun Grid Engine options valid with *qsub(1)* or *qrsh(1)* can be specified with *qmake* - see *submit(1)* for a description of all Sun Grid Engine commandline options. The *make(1)* manual page describes the *gmake* commandline syntax.

The syntax of *qmake* makefiles corresponds to *gmake* and is described in the "GNU Make Manual".

EXAMPLES

```
qmake -pe compiling 1-10 -
```

will request between 1 and 10 slots in parallel environment "compiling" on the same architecture as the submit host. The *make* tasks will inherit the complete environment of the calling shell. It will execute as many parallel tasks as slots have been granted by Sun Grid Engine.

```
qmake -- -j 4
```

will request between 1 and 4 slots in parallel environment "make" on the same architecture as the submit host.

```
qmake -l arch=solaris -pe make 3
```

will request 3 parallel *make* tasks to be executed on hosts of architecture "solaris". The submit may be done on a host of any architecture.

The shell script:

```
#!/bin/sh
qmake -inherit --
```

can be submitted by:

```
qsub -pe make 1-10 [further_codine_options] <script>
```

Qmake will inherit the resources granted for the job submitted above under parallel environment "make".

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of /usr/CODINE is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qmake* uses (in the order of precedence):

- The name of the cell specified in the environment variable **COD_CELL**, if it is set.

- The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qmake* client resides. Per default the local host is used.

KNOWN PROBLEMS

Slow NFS server

Very low file server performance may lead to problems on depending files.

Example: Host a compiles a.c to a.o, host b compiles b.c to b.o, host c shall link program c from a.o and b.o. In case of very bad NFS performance, host c might not yet see files a.o and b.o.

Multiple commands in one rule

If multiple commands are executed in one rule, the makefile has to ensure that they are handled as one commandline.

Example:

```
libx.a:  
    cd x  
    ar ru libx.a x.o
```

Building libx.a will fail, if the commands are executed in parallel (and possibly on different hosts). Write the following instead:

```
libx.a:  
    cd x ; ar ru libx.a x.o
```

or

```
libx.a:  
    cd x ; \  
    ar ru libx.a x.o
```

SEE ALSO

submit(1) as well as *make(1)* (GNU make manpage) and *The GNU Make Manual* in `<cod_root>/3rd_party/qmake`.

COPYRIGHT

Qmake contains portions of Gnu Make (*gmake*), which is the copyright of the Free Software Foundation, Inc., Boston, MA, and is protected by the Gnu General Public License.

See *sge_intro(1)* and the information provided in `<cod_root>/3rd_party/qmake` for a statement of further rights and permissions.

NAME

qmod – modify a Sun Grid Engine queue

SYNTAX

qmod [**options**] [**job/task_id_list** | **queue_list**]

DESCRIPTION

Qmod enables users classified as *owners* (see *queue_conf(5)* for details) of a workstation to modify the state of Sun Grid Engine queues for his/her machine as well as the state of his/her own jobs. A manager/operator or root can execute *qmod* for any queue and job in a cluster.

OPTIONS

-c

Clears the error state of the specified queue(s).

-d

Disables the queue(s), i.e. no further jobs are dispatched to disabled queues while jobs already executing in these queues are allowed to finish.

(Is the successor of the Sun Grid Engine (CODINE) version 3 -soc option.)

-e

Enables the queue(s).

(Is the successor of the Sun Grid Engine (CODINE) version 3 -xsoc option.)

-f

Force the modification action for the queue despite the apparent current state of the queue. For example if a queue appears to be suspended but the job execution seems to be continuing the manager/operator can force a suspend operation which will send a SIGSTOP to the jobs. In any case, the queue or job status will be set even if the *cod_execd(8)* controlling the queues/jobs cannot be reached. Requires manager/operator privileges.

–help

Prints a listing of all options.

–s

If applied to queues, suspends the queues and any jobs which might be active. If applied to running jobs, suspends the jobs. If a job is both suspended explicitly and via suspension of its queue, a following unuspend of the queue will not release the suspension state on the job.

–us

If applied to queues, unuspends the queues and any jobs which might be active. If applied to jobs, unuspends the jobs. If a job is both suspended explicitly and via suspension of its queue, a following unuspend of the queue will not release the suspension state on the job.

–verify

performs no modifications but just prints what would be done if **–verify was not present**.

job/task_id_list | queue_list

The jobs or queues upon which *qmod* is supposed to operate. The **job/task_id_list** is specified by one of the following forms:

job_id[.task_range][,job_id[.task_range],...]

job_id[.task_range][job_id[.task_range] ...]

If present, the *task_range* restricts the effect of the *qmod* operation to the job array task range specified as suffix to the job id (see the **–t** option to *qsub(1)* for further details on job arrays).

The task range specifier has the form *n[-m[:s]][,n[-m[:s]], ...]* or *n[-m[:s]][n[-m[:s]] ...]* and thus consists of a comma or blank separated list of range specifiers *n[-m[:s]]*. The ranges are concatenated to the complete task id range. Each range may be a single number, a simple range of the form *n-m* or a range with a step size.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qmod* uses (in the order of precedence):

The name of the cell specified in the environment

variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qmod* client resides. Per default the local host is used.

FILES

<cod_root>/<cell>/common/act_qmaster
Sun Grid Engine master host file

SEE ALSO

sge_intro(1), *sge_ckpt(1)*, *qstat(1)*, *queue_conf(5)*, *cod_execd(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`qmon` – X-Windows OSF/Motif graphical user's interface for Sun Grid Engine

SYNTAX

`qmon` [options]

DESCRIPTION

Qmon allows administrators and users to manipulate the Sun Grid Engine system from an X-Window environment. *Qmon* provides various dialogues linked together in multiple ways. For each task the user wishes to accomplish via *qmon* a corresponding dialogue is provided. There are multiple ways to address the proper dialogue for a certain task:

- ❑ The *qmon* main window that comes up first on start-up contains icon buttons for all major administrative and user tasks. A functionality tooltip is displayed when pointing at the different icons.
- ❑ A **Task** pulldown menu button appears in the *qmon* main window menu bar. Clicking on it opens a list of available tasks. Selecting one of them opens the corresponding dialogue.
- ❑ The **Task** pulldown menu also contains the key accelerators which can be used to invoke the task dialogues directly from the main window by pressing a certain button sequence on the keyboard.
- ❑ While navigating through a certain dialogue and its dialogue subhierarchy, links to other dialogues occur whenever a connection between both dialogues is obvious. Pushing the buttons that identify the links opens up the other dialogues.

OPTIONS

The supported options are the standard X Toolkit options as described in *X(1)* section **Options**. Furthermore, *qmon* supports:

–cmap

Installs a private color map for *qmon*. This is sometimes useful if other applications have already allocated lots of colors and if *qmon*, therefore, prints corresponding error messages.

Note – Using a private color map, however, will result in color map switches whenever you enter or leave *qmon* windows.

-fontFamily {big|medium|small}

Notifies *qmon* to use different sized font families for different resolution sizes.

-help

Displays usage information.

-nologo

Startup without logo.

DIALOGUES

Job Control

The **Job Control** dialogue provides a folder of tabulated lists of the still pending jobs, already running jobs and recently finished jobs. The dialogue allows for detailed information on the jobs as well as for the deletion and suspension of jobs being selected. In addition the job control dialogue offers links to the **Submit** dialogue in order to submit new jobs or to change attributes of pending jobs (**Qalter** button). The shown displayed fields in the tabular display and the jobs displayed can be customized by pressing the **Customize** button. This customization can be saved to the *~/qmon_preferences* file and is used on following startups for the initial configuration of the **Job Control** dialogue.

Queue Control

The **Queue Control** dialogue with its sub-dialogue hierarchy enables the user to control the status of the Sun Grid Engine queues being actually configured in the system and allows the administrator to add new queues or to modify or delete already existing ones. Each icon button in the top level **Queue Control** dialogue window represents a configured Sun Grid Engine queue. The icon symbols, the coloring and the text on the buttons informs about the architecture, the status and some basic attributes of the queues. The top level dialogue also allows for deleting those queues previously selected. Queues are selected by clicking with the left mouse button on the icons or into a rectangular area surrounding the buttons.

By pushing the **Add** or **Modify** button or using a pop-up menu that is raised when clicking the right mouse button in the icon window of the top level **Queue Control** dialogue, a sub-dialogue for configuring Sun Grid Engine queues is opened. A queue needs to be selected to use the modify operation. The configuration sub-dialogue allows for definition of the queue and host name or displays the corresponding names in case of a modification. The queue configuration parameters (see *queue_conf(5)*) are subdivided in different categories (**General Configuration, Execution Methods, Checkpointing, Load/Suspend Thresholds, Limits, Complexes, User Access, Project Access** (only for Sun Grid

Engine, Enterprise Edition), Subordinate Queues, Owners) which are selectable by the tab widget area presented in the lower region of the queue configuration sub-dialogue. The administrator may select default values from already configured queues (**Clone** button). By pushing the **Ok** button, the definitions are registered with *cod_qmaster(8)*. The **Queue Control** dialogue can be customized in a similar way as the **Job Control** dialogue. The settings applied here are also saved in *~/qmon_preferences*.

Submit

The **Job Submission** dialogue serves for submitting batch and interactive jobs and is also invoked when changing attributes of pending jobs from the **Job Control** dialogue explained above (**Qalter** button). To toggle between batch and interactive jobs please use the **Batch/Interactive** button at the top of the button column on the right side of the **Job Submission** screen.

The dialogue consists of a folder containing two job preparation dialogue pages. The most frequently used parameters in the course of a job submission are offered on the **General** page. A job script has to be defined, all other fields are optional. If the job demands for specification of advanced requirements, the **Advanced** tab can be used to switch to an enhanced parameter display.

If resource requirements are mandatory for the job, the **Request Resources** icon button has to be used to pop up the **Requested Resources** sub-dialogue. This sub-dialogue allows for selection of the required resources of the job and for definition of the quantities in which this resources are to be provided. The **Available Resources** are constituted by those complex attributes being declared *requestable* (see *complex(5)* for details). Resource requirements can be made **Hard**, i.e. they must be met before a job can be started in a queue, or **Soft**, i.e. they are granted on an as available basis.

Closing the **Requested Resources** sub-dialogue with the done button books the specified requirement for the job. Pushing the **Submit** button on the top level **Submit** dialogue submits the job.

Complex Config

The **Complex Config** allows the administrator to add new complexes or to modify or delete existing ones (see *complex(5)*). The dialogue offers a selection list for the existing complexes and displays the configuration of the one being selected. By pushing the **Delete** button, the selected complex is deleted from the configuration. Pushing the **Add/Modify** button will open a complex configuration dialogue, which allows to create new complexes or which provides the means to change the existing ones. If a new complex is to be created, a name must be defined for it. The name of the complex to be modified is displayed in the same text input field in case of a modify operation. The complex configuration dialogue provides a tabulated list of the complex entries and an input region for the

declaration of new or modified entries. The **Add** button updates the tabulated list with the new or changed entry and the **Ok** button registers the additional or modified complex with *cod_qmaster(8)*.

Host Config

Three types of host lists can be maintained via the **Host Config** dialogue:

Administration Hosts

Submit Hosts

Execution Hosts

The host list to be manipulated is selected via clicking at one of the tabs named correspondingly. The first two host lists only provide for adding or deleting entries, thereby allowing administrative or submit permission for the hosts on the lists, or denying it otherwise respectively. The execution host list entries in addition provide the ability to define scaling factors for the load sensors, consumable complex attributes and access attributes (access, xaccess and projects, xprojects for Sun Grid Engine, Enterprise Edition mode only) as described in *complex(5)*. In a Sun Grid Engine, Enterprise Edition system CPU, memory and I/O usage reported for running jobs can be scaled in addition and the relative performance of a host can be define with the **Resource Capability Factor** (see *host_conf(5)*).

Cluster Config

This dialogue maintains the cluster global configuration as well as host specific derivatives (see *sge_conf(5)*). When opened, the dialogue displays a selection list for all hosts which have a configuration assigned. The special name “global” refers to the cluster global configuration. By pushing the **Add/Modify** button a sub-dialogue is opened, which allows for modification of the cluster configuration. For host specific configurations the ‘global’ host specific configuration fields are set insensitive and only the allowed parameters can be manipulated.

Scheduler Config

The **Scheduler Configuration** dialogue provides the means to change the behavior of the Sun Grid Engine scheduler daemon *cod_schedd(8)*. The dialogue contains a representation for all scheduler configuration parameters as described in *sched_conf(5)*. It is subdivided in the two sections **General Parameters** and **Load Adjustments** which can be selected via the corresponding tabs. The **Ok** button registers any changes with *cod_qmaster(8)*.

Calendar Config

The **Calendar Config** allows the administrator to add new calendars or to modify or delete existing ones (see *calendar_conf(5)*). The dialogue offers a selection list for the existing calendars and displays the configuration of the one being selected. By pushing the **Delete** button, the selected calendar is deleted from the configuration. Pushing the **Add/Modify** button will open a calendar configuration dialogue, which allows to create new calendars or which provides the means to change the existing ones. The **Ok** button registers the additional or modified calendar with *cod_qmaster(8)*.

User Config

User permissions are controlled via the **User Config** dialogue. The tab widget in the left section of the dialogue allows for selecting between

Configuration of **Manager** accounts.

Configuration of **Operator** accounts.

Definition of **Usersets**.

Definition of **User** accounts (Sun Grid Engine, Enterprise Edition mode only).

Those user accounts added to the list of manager or operator accounts are given permission to act as managers or operators respectively when accessing Sun Grid Engine under their own account.

The userset lists are used together with the **user_lists** and **xuser_lists** host, queue, project and cluster configuration parameters (see *queue_conf(5)*, *project(5)* and *sge_conf(5)*) to control access of users to hosts, queues, projects (only available in a Sun Grid Engine, Enterprise Edition system) and the entire cluster. A userset is just a collection of user names and UNIX group names. Group names are identified by prefixing them with a "@" sign. The already defined usersets are displayed in a selection list. These lists can be modified and new lists can be created using the **Userset** definition dialogue.

In a Sun Grid Engine, Enterprise Edition system usersets can be used as **Access List** (equivalent to their usage in a Sun Grid Engine system) and/or as **Department** required for the so called **Functional Policy** and **Override Policy** (see **Ticket Config** below).

A Sun Grid Engine, Enterprise Edition system also requires adding accounts having access to the system as entries to the Sun Grid Engine, Enterprise Edition user database (see *user(5)*) This can be done with the **User** sub-dialogue.

The **Tickets** button in the button list on the right side of the dialogue opens the **Ticket Config** dialogue (see below). This is also only available in a Sun Grid Engine, Enterprise Edition system.

PE Config

Parallel environment (PE) interfaces can be configured with this dialogue. PE interfaces are necessary to describe the way how parallel programming environments like PVM (Parallel Virtual Machine), MPI (Message Passing Interface) or shared memory parallel systems are to be instantiated and to impose access restrictions onto the PEs. When the dialogue is opened a list of the already configured PEs is displayed together with the current configuration (see *pe_conf(5)*) of the selected PE interface. To add new PE interfaces or to modify existing ones, an **Add** and a **Modify** button is available which opens a PE interface configuration sub-dialogue. After applying the changes and quitting this sub-dialogue with the **OK** button, the new or modified PE interface is registered with *cod_qmaster(8)*.

Checkpoint Config

Checkpointing environment interfaces can be configured with this dialogue. Checkpointing environments are necessary to describe the attributes which the different checkpointing methods and their derivatives on various operating system platforms supported by Sun Grid Engine have. When the dialogue is opened a list of the already configured checkpointing environments is displayed together with the current configuration (see *checkpoint(5)*) of the selected checkpointing environment. To add new checkpointing environment or to modify existing ones, an **Add** and a **Modify** button is available which opens a checkpointing environment configuration sub-dialogue. After applying the changes and quitting this sub-dialogue with the **OK** button, the new or modified checkpointing environment is registered with *cod_qmaster(8)*.

Ticket Conf

This dialogue offers an overview and editing screen for allocating tickets to the share-based, functional and override scheduling policies. It is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

The **Deadline Job** button opens the **User Conf** dialogue box. Please change to the Userset sub-dialogue and select the userset named “deadlineusers”. Only users of this userset may submit deadline jobs.

The **Share Tree Policy** button opens the dialogue for creating and editing the Sun Grid Engine, Enterprise Edition share tree (see *share_tree(5)* and *schedd_conf(5)* for a description of the configuration parameters).

The **Functional Policy** button opens the dialogue for creating and editing the allocation of the functional shares (see *sched_conf(5)*, *access_list(5)*, *project(5)*, *queue_conf(5)* and *user(5)* for a description of the different types of functional shares and the configurable weighting parameters).

The **Override Policy** button opens the dialogue for creating and editing the allocation of override tickets (see *access_list(5)*, *project(5)*, *queue_conf(5)* and *user(5)* for a description of the different types of override tickets).

Project Conf

This button opens a dialogue for creating projects. It is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

The dialogue offers a selection list for the existing projects and displays the configuration of the one being selected. By pushing the **Delete** button, the selected project is deleted from the configuration. Pushing the **Add/Modify** button will open a project configuration dialogue, which allows to create new projects or which provides the means to change the existing ones. Project configuration in essence means giving or denying access to a project for usersets (see **User Conf** above as well as *project(5)*). The Ok button registers the additional or modified project with *cod_qmaster(8)*.

Browser

The **Object Browser** dialogue's purpose is manifold: First of all, Sun Grid Engine and *qmon* messages such as notification of error or success concerning a previously taken action can be displayed in the dialogue's output window. Also the standard output and the standard error output of *qmon* can be diverted to the **Object Browser** output window.

Additionally the **Object Browser** can be used to display continuous information about Sun Grid Engine objects as the mouse pointer moves over their representation as icons or table entries in other *qmon* dialogues. Currently, only the display of the configuration of two Sun Grid Engine objects in two separate dialogues is supported:

- Queue configurations are displayed as soon as the mouse pointer enters a queue icon in the top level **Queue Control** dialogue (see above). This facility is activated by pushing the **Queue** button in the **Object Browser** dialogue.
- Detailed job information is printed as soon as the user moves the mouse pointer over a line in the **Job Control** dialogue (see above) being assigned to a running or pending job.
- Additionally job scheduling information is displayed in the browser if the **Why ?** button in the **Job Control** dialogue is pressed. In this case the Browser dialogue is opened implicitly and any scheduling related information is displayed.

Exit

The **Exit** icon button is not linked with a dialogue. Its sole purpose is to close all active *qmon* dialogues and to exit the application.

RESOURCES

The available resources, their meaning and the syntax to be followed in order to modify them are described in the default *qmon* resource file (see the section **Files** below for the location of the resource file).

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of `/usr/CODINE` is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qmon* uses (in the order of precedence):

The name of the cell specified in the environment
variable `COD_CELL`, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qmon* client resides. Per default the local host is used.

RESTRICTIONS

If the line to be entered in an editing window is longer than the width of the window, then the text just runs off the end of the window.

FILES

`<codine_root>/qmon/Qmon`

Qmon sample resources file

`/usr/lib/X11/defaults/Qmon`

Qmon system resources file

`$HOME/Qmon`

Qmon user resources file

`$HOME/.qmon_preferences`

Qmon job/queue customization file

SEE ALSO

sgc_intro(1), *sgc_conf(5)*, *access_list(5)*, *sgc_pe(5)*, *calendar_conf(5)*, *complex(5)*, *project(5)*, *queue_conf(5)*, *sched_conf(5)*, *user(5)*, *cod_qmaster(8)*.

COPYRIGHT

See *sgc_intro(1)* and the information provided in `<cod_root>/3rd_party/qmon` for a statement of further rights and permissions and for credits to be given to public domain and freeware widget developers.

NAME

qrls – release Sun Grid Engine jobs from previous hold states

SYNTAX

qrls [**-h** {ulols},...] [**-help**] [**job/task_id_list**]

qrls [**-h** {ulols},...] [**-help**] **-u** **user_list** | **-uall**

DESCRIPTION

Qrls provides a means for a user/operator/manager to release so called *holds* from one or more jobs pending to be scheduled for execution. As long as any type of hold is assigned to a job, the job is not eligible for scheduling.

Holds can be assigned to jobs with the *qhold(1)*, *qsub(1)* or the *qalter(1)* command.

There are three different types of holds:

user

User holds can be assigned and removed by managers, operators and the owner of the jobs.

operator

Operator holds can be assigned and removed by managers and operators.

system

System holds can be assigned and removed by managers only.

If no hold type is specified with the **-h** option (see below) the user hold is assumed by default.

An alternate way to release holds is the *qalter(1)* command (see the **-h** option).

OPTIONS

-h {ulols},...

Releases a u(ser), o(perator) or s(system) hold or a combination thereof from one or more jobs.

-help

Prints a listing of all options.

-u username,... | -uall

Modifies the hold state of those jobs which were submitted by users specified in the list of usernames. For managers it is possible to use the **qrls -uall** command to modify the hold state for jobs of all users.

If you use the **-u** or **-uall** switch it is permitted to specify an additional *job/task_id_list*.

job/task_id_list

Specified by the following form:

job_id[.task_range][,job_id[.task_range],...]

If present, the *task_range* restricts the effect of the operation to the job array task range specified as suffix to the job id (see the **-t** option to *qsub(1)* for further details on job arrays).

The task range specifier has the form n[-m[:s]]. The range may be a single number, a simple range of the form n-m or a range with a step size.

Instead of *job/task_id_list* it is possible to use the keyword 'all' to modify all jobs of the current user.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of /usr/CODINE is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qrls* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qrls* client resides. Per default the local host is used.

SEE ALSO

sgc_intro(1), *qalter(1)*, *qhold(1)*, *qsub(1)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

`qselect` – select queues.

SYNTAX

```
qselect [ -help ] [ -l resource=val,... ] [ -pe pe_name,... ]  
          [ -q queue,... ] [ -U user,... ]
```

DESCRIPTION

qselect prints a list of Sun Grid Engine queue names corresponding to selection criteria specified in the *qselect* arguments described below. The output of *qselect* can be fed into other Sun Grid Engine commands to apply actions on the selected queue sets. For example together with the *-mqattr* option to *qconf(1)*, *qselect* can be used to modify queue attributes on a set of queues.

OPTIONS

-help

Prints a listing of all options.

-l resource[=value],...

Defines the resources to be granted by the queues which should be included in the queue list output.

-pe pe_name,...

Includes queues into the output which are attached to at least one of the parallel environments enlisted in the comma separated option argument.

-q queue,...

Directly specifies the queues to be included in the output. This option usually is only meaningful in conjunction with another *qselect* option to extract a subset of queue names from a list given by **-q**.

-U user,...

Includes the queues to which the specified users have access in the *qselect* output.

EXAMPLES

```
qselect -l arch=linux
qselect -l arch=linux -U andreas,shannon
qconf -mqattr h_vmem=1GB 'qselect -l arch=linux'
```

The first example prints the names of those queues residing on Linux machines. The second command in addition restricts the output to those queues with access permission for the users *andreas* and *shannon*. The third command changes the queue attribute *h_vmem* to 1 Gigabyte on queues residing on Linux machines (see the *qconf(1)* manual page for details on the *-mqattr* option and the *queue_conf(5)* manual page on details of queue configuration entries).

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qselect* uses (in the order of precedence):

- The name of the cell specified in the environment variable **COD_CELL**, if it is set.

- The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qselect* client resides. Per default the local host is used.

FILES

<codine_root>/<cell>/common/act_qmaster
Sun Grid Engine master host file

SEE ALSO

sgc_intro(1), *qconf(1)*, *qmod(1)*, *qstat(1)*, *queue_conf(5)*, *cod_commd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

qstat – show the status of Sun Grid Engine jobs and queues

SYNTAX

```
qstat [ -ext ] [ -f ] [ -F [resource_name,...] ] [ -g d ] [ -help ]
      [ -j [job_list] ] [ -l resource=val,... ] [ -ne ]
      [ -pe pe_name,... ] [ -q queue,... ] [ -r ]
      [ -s {rlplslzlhlholhslhjlhalh}[+] ] [ -t ] [ -U user,... ]
      [ -u user,... ]
```

```
qstat [ -f ] -qstd [ hostname ]
```

DESCRIPTION

qstat shows the current status of the available Sun Grid Engine queues and the jobs associated with the queues. Selection options allow you to get information about specific jobs, queues or users. Without any option *qstat* will display only a list of jobs with no queue status information.

In the second form *qstat* displays the status of the *Queueing System Transfer Daemons* (see *cod_qstd(8)*) currently in operation. There are only *cod_qstds* running if the Sun Grid Engine *Queueing System Interface (QSI)* is licensed and properly installed. Please refer to the *Sun Grid Engine Installation and Administration Guide* for detailed information.

OPTIONS

–alarm

Displays the reason(s) for queue alarm states. Outputs one line per reason containing the resource value and threshold. For details about the resource value please refer to the description of the **Full Format** in section **OUTPUT FORMATS** below.

–ext

This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Displays additional Sun Grid Engine, Enterprise Edition relevant information for each job (see **OUTPUT FORMATS** below).

-f

Specifies a “full” format display of information. The **-f** option causes summary information on all queues to be displayed along with the queued job list.

-F [resource_name,...]

Like in the case of **-f** information is displayed on all jobs as well as queues. In addition, *qstat* will present a detailed listing of the current resource availability per queue with respect to all resources (if the option argument is omitted) or with respect to those resources contained in the *resource_name* list. Please refer to the description of the **Full Format** in section **OUTPUT FORMATS** below for further detail.

-g d

Displays job arrays verbosely in a one line per job task fashion. By default, job arrays are grouped and all tasks with the same status (for pending tasks only) are displayed in a single line. The job array task id range field in the output (see section **OUTPUT FORMATS**) specifies the corresponding set of tasks.

The **-g** switch currently has only the single option argument **d**. Other option arguments are reserved for future extensions.

-help

Prints a listing of all options.

-j [job_list]

Prints either for all pending jobs or the jobs contained in *job_list* the reason for not being scheduled.

-l resource[=value],...

Defines the resources required by the jobs or granted by the queues on which information is requested. Matching is performed on queues. The pending jobs are restricted to jobs that might run in one of the above queues.

-ne

In combination with **-f** the option suppresses the display of empty queues. This means all queues where actually no jobs are running are not displayed.

-pe pe_name,...

Displays status information with respect to queues which are attached to at least one of the parallel environments enlisted in the comma separated option argument. Status information for jobs is displayed either for those which execute in one of the selected queues or which are pending and might get scheduled to those queues in principle.

-q queue,...

Specifies the queue to which job information is to be displayed.

-r

Prints extended information about the resource requirements of the displayed jobs. Please refer to the **OUTPUT FORMATS** sub-section **Expanded Format** below for detailed information.

-s {plrslzlhulholhslhjlhalh}[+]

Prints only jobs in the specified state, any combination of states is possible. **-s prs** corresponds to the regular *qstat* output without **-s** at all. To show recently finished jobs, use **-s z**. To display jobs in user/operator/system hold, use the **-s hu/ho/hs** option. The **-s ha** option shows jobs which were submitted with the *qsub -a* command. *qstat -s hj* displays all jobs which are not eligible for execution unless the job has entries in the job dependency list. (see **-a** and **-hold_jid** option to *qsub(1)*).

-t

Prints extended information about the controlled sub-tasks of the displayed parallel jobs. Please refer to the **OUTPUT FORMATS** sub-section **Expanded Format** below for detailed information. Sub-tasks of parallel jobs should not be confused with job array tasks (see **-g** option above and **-t** option to *qsub(1)*).

-U user,...

Displays status information with respect to queues to which the specified users have access. Status information for jobs is displayed either for those which execute in one of the selected queues or which are pending and might get scheduled to those queues in principle.

-u user,...

Display information only on those jobs and queues being associated with the users from the given user list. Queue status information is displayed if the **-f** or **-F** options are specified additionally and if the user runs jobs in those queues.

-qstd [hostname]

Display the status of other queueing systems configured to be interfaced by Sun Grid Engine. Without the optional *hostname* *qstat* displays information on all transfer queues and the corresponding hosts. If the *hostname* is present, the information provided only refers to that host.

If an additional **-f** switch is provided, *qstat* lists a rather complete set of information about the transfer queue(s) and the corresponding host(s). If the **-f** switch is absent, the status listing only contains information about the jobs having been forwarded to the other queueing systems by Sun Grid Engine. This option is only operational if the Sun Grid Engine queueing system interface is licensed and properly installed. Please ask your system administrator.

OUTPUT FORMATS

Depending on the presence or absence of the **-alarm**, **-f** or **-F** and **-r** and **-t** option three output formats need to be differentiated. PP In case of a Sun Grid Engine, Enterprise Edition system, the **-ext** option may be used to display additional information for each job.

Reduced Format (without **-f and **-F**)**

Following the header line a line is printed for each job consisting of

- the job ID.

- the priority of the jobs as assigned to them via the **-p** option to *qsub(1)* or *qalter(1)* determining the order of the pending jobs list.
- the name of the job.
- the user name of the job owner.
- the status of the job – one of t(ransferring), r(unning), R(estarted), s(uspended), S(uspended), T(hreshold), w(aiting) or h(old).

The states t(ransferring) and r(unning) indicate that a job is about to be executed or is already executing, whereas the states s(uspended), S(uspended) and T(hreshold) show that an already running jobs has been suspended. The s(uspended) state is caused by suspending the job via the *qmod(1)* command, the S(uspended) state indicates that the queue containing the job is suspended and therefore the job is also suspended and the T(hreshold) state shows that at least one suspend threshold of the corresponding queue was exceeded (see *queue_conf(5)*) and that the job has been suspended as a consequence. The state R(estarted) indicates that the job was restarted. This can be caused by a job migration or because of one of the reasons described in the -r section of the *qsub(1)* command.

The states w(aiting) and h(old) only appear for pending jobs. The h(old) state indicates that a job currently is not eligible for execution due to a hold state assigned to it via *qhold(1)*, *qalter(1)* or the *qsub(1)* **-h** option or that the job is waiting for completion of the jobs to which job dependencies have been assigned to the job via the **-hold_jid** option of *qsub(1)* or *qalter(1)*.

- the submission or start time and date of the job.
- the queue the job is assigned to (for running or suspended jobs only).
- the function of the running jobs (MASTER or SLAVE – the latter for parallel jobs only).
- the job array task id. Will be empty for non-array jobs. See the **-t** option to *qsub(1)* and the **-g** above for additional information.

If the **-t** option is supplied, each job status line also contains

- the parallel task ID (do not confuse parallel tasks with job array tasks),
- the status of the parallel task – one of r(unning), R(estarted), s(uspended), S(uspended), T(hreshold), w(aiting), h(old), or x(exited).
- the cpu, memory, and I/O usage (Sun Grid Engine, Enterprise Edition only),
- the exit status of the parallel task,
- and the failure code and message for the parallel task.

Full Format (with **-f** and **-F**)

Following the header line a section for each queue separated by a horizontal line is provided. For each queue the information printed consists of

- the queue name,
- the queue type – one of B(atch), I(nteractive), C(heckpointing), P(arallel), T(ransfer) or combinations thereof,
- the number of used and available job slots,

- the load average of the queue host,
- the architecture of the queue host and
- the state of the queue – one of u(nknown) if the corresponding *cod_execd(8)* cannot be contacted, a(larm), A(larm), C(alendar suspended), s(uspended), S(ubordinate), d(isabled), D(isabled), E(rror) or combinations thereof.

If the state is a(larm) at least one of the load thresholds defined in the *load_thresholds* list of the queue configuration (see *queue_conf(5)*) is currently exceeded, which prevents from scheduling further jobs to that queue.

As opposed to this, the state A(larm) indicates that at least one of the suspend thresholds of the queue (see *queue_conf(5)*) is currently exceeded. This will result in jobs running in that queue being successively suspended until no threshold is violated.

The states s(uspended) and d(isabled) can be assigned to queues and released via the *qmod(1)* command. Suspending a queue will cause all jobs executing in that queue to be suspended.

The states D(isabled) and C(alendar suspended) indicate that the queue has been disabled or suspended automatically via the calendar facility of Sun Grid Engine (see *calendar_conf(5)*), while the S(ubordinate) state indicates, that the queue has been suspend via subordination to another queue (see *queue_conf(5)* for details). When suspending a queue (regardless of the cause) all jobs executing in that queue are suspended too.

If an E(rror) state is displayed for a queue, *cod_execd(8)* on that host was unable to locate the *cod_shepherd(8)* executable on that host in order to start a job. Please check the error logfile of that *cod_execd(8)* for leads on how to resolve the problem. Please enable the queue afterwards via the **-c** option of the *qmod(1)* command manually.

If the **-F** option was used, resource availability information is printed following the queue status line. For each resource (as selected in an option argument to **-F** or for all resources if the option argument was omitted) a single line is displayed with the following format:

- a one letter specifier indicating whether the current resource availability value was dominated by either
- 'g' - a cluster global,
- 'h' - a host total or
- 'q' - a queue related resource consumption.
- a second one letter specifier indicating the source for the current resource availability value, being one of
- 'l' - a load value reported for the resource,
- 'L' - a load value for the resource after administrator defined load scaling has been applied,

- 'c' - availability derived from the consumable resources facility (see *complexes(5)*), 'v' - a default complexes configuration value never overwritten by a load report or a consumable update or
- 'f' - a fixed availability definition derived from a non-consumable complex attribute or a fixed resource limit.
- after a colon the name of the resource on which information is displayed.
- after an equal sign the current resource availability value.

The displayed availability values and the sources from which they derive are always the minimum values of all possible combinations. Hence, for example, a line of the form "qf:h_vmem=4G" indicates that a queue currently has a maximum availability in virtual memory of 4 Gigabyte, where this value is a fixed value (e.g. a resource limit in the queue configuration) and it is queue dominated, i.e. the host in total may have more virtual memory available than this, but the queue doesn't allow for more. Contrarily a line "hl:h_vmem=4G" would also indicate an upper bound of 4 Gigabyte virtual memory availability, but the limit would be derived from a load value currently reported for the host. So while the queue might allow for jobs with higher virtual memory requirements, the host on which this particular queue resides currently only has 4 Gigabyte available.

If the **-alarm** option was used, information about resources is displayed, that violate load or suspend thresholds.

The same format as with the **-F** option is used with following extensions:

- the line starts with the keyword 'alarm'
- appended to the resource value is the type and value of the appropriate threshold

After the queue status line (in case of **-f**) or the resource availability information (in case of **-F**) a single line is printed for each job running currently in this queue. Each job status line contains

- the job ID,
- the job name,
- the job owner name,
- the status of the job – one of t(ransferring), r(unning), R(estarted), s(uspended), S(uspended) or T(hreshold) (see the **Reduced Format** section for detailed information),
- the start date and time and the function of the job (MASTER or SLAVE - only meaningful in case of a parallel job) and
- the priority of the jobs.

If the **-t** option is supplied, each job status line also contains

- the task ID,
- the status of the task – one of r(unning), R(estarted), s(uspended), S(uspended), T(hreshold), w(aiting), h(old), or x(exited) (see the **Reduced Format** section for detailed information),
- the cpu, memory, and I/O usage (Sun Grid Engine, Enterprise Edition only),
- the exit status of the task,

and the failure code and message for the task.

Following the list of queue sections a *PENDING JOBS* list may be printed in case jobs are waiting for being assigned to a queue. A status line for each waiting job is displayed being similar to the one for the running jobs. The differences are that the status for the jobs is w(aiting) or h(old), that the submit time and date is shown instead of the start time and that no function is displayed for the jobs.

In very rare cases, e.g. if *cod_qmaster(8)* starts up from an inconsistent state in the job or queue spool files or if the **clean queue (-cq)** option of *qconf(1)* is used, *qstat* cannot assign jobs to either the running or pending jobs section of the output. In this case as job status inconsistency (e.g. a job has a running status but is not assigned to a queue) has been detected. Such jobs are printed in an *ERROR JOBS* section at the very end of the output. The *ERROR JOBS* section should disappear upon restart of *cod_qmaster(8)*. Please contact your Sun Grid Engine support representative if you feel uncertain about the cause or effects of such jobs.

Expanded Format (with -r)

If the **-r** option was specified together with *qstat*, the following information for each displayed job is printed (a single line for each of the following job characteristics):

- The hard and soft resource requirements of the job as specified with the *qsub(1)* **-l** option.
- The requested parallel environment including the desired queue slot range (see **-pe** option of *qsub(1)*).
- The requested checkpointing environment of the job (see the *qsub(1)* **-ckpt** option).
- In case of running jobs, the granted parallel environment with the granted number of queue slots.

Enhanced Sun Grid Engine, Enterprise Edition Output (with -ext)

For each job the following additional items are displayed:

project

The project to which the job is assigned as specified in the *qsub(1)* **-P** option.

department

The department, to which the user belongs (use the **-sul** and **-su** options of *qconf(1)* to display the current department definitions).

deadline

The deadline initiation time of the job as specified with the *qsub(1)* **-dl** option.

cpu

The current accumulated CPU usage of the job.

mem

The current accumulated memory usage of the job.

io

The current accumulated IO usage of the job.

tckts

The total number of tickets assigned to the job currently

ovrts

The override tickets as assigned by the **-ot** option of *qalter(1)*.

otckt

The override portion of the total number of tickets assigned to the job currently

dtckt

The deadline portion of the total number of tickets assigned to the job currently

ftckt

The functional portion of the total number of tickets assigned to the job currently

stckt

The share portion of the total number of tickets assigned to the job currently

share

The share of the total system to which the job is entitled currently.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qstat* uses (in the order of precedence):

The name of the cell specified in the environment

variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qstat* client resides. Per default the local host is used.

FILES

<codine_root>/<cell>/common/act_qmaster
Sun Grid Engine master host file

SEE ALSO

sge_intro(1), *qalter(1)*, *qconf(1)*, *qhold(1)*, *qhost(1)*, *qmod(1)*, *qsub(1)*, *queue_conf(5)*, *cod_commd(8)*, *cod_execd(8)*, *cod_qmaster(8)*, *cod_qstd(8)*, *cod_shepherd(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

qtcssh – tcsh v6.09 with transparent remote execution by use of qrsh.

SYNTAX

qtcssh [**tcsh options** | **-ABLR**]

DESCRIPTION

Qtcssh is an extension to the popular *csh(1)* derivative *tcsh*. It allows the transparent remote execution commands entered in *qtcssh* controlled via Sun Grid Engine. *Qtcssh* can be used as interactive command interpreter as well as for the processing of *tcsh* shell scripts.

When invoked, *qtcssh* identifies which commands are to be run remotely and which are not. For this purpose the files `<cod_root>/<cell>/common/qttask` and `~/qttask` are processed. Each line in these files defines a command that is intended to be run remotely (see *qttask(5)* for a definition of the file format). The `.qttask` file in the user's home directory contains the user's remote task specification, while the file in the common directory is maintained by the administrator and defines a cluster-wide default behavior. The contents of the administrator supplied *qttask(5)* file are completely overridden in case there is an appropriate entry in the users *qttask(5)* file. This is prevented in case an exclamation mark is prefixed to the command name in the administrators *qttask* file.

Qtcssh always attempts to start the designated tasks remotely via *qrsh(1)*. Exceptions are

- ❑ if the user enters such commands via a relative or absolute pathname instead of the stand-alone command name (see *qttask(5)* for more information).
- ❑ if the environment variable **JOB_ID** is set and thus *qtcssh* assumes that execution already happens remotely within a Sun Grid Engine job and thus executes tasks locally. This avoids unwanted recursions but can be overridden by the command-line option **-R** and the built-in command *qrshmode -R* (see corresponding descriptions below).
- ❑ if *qtcssh* cannot establish a connection of Sun Grid Engine during start-up. This allows to use *qtcssh* as login shell without the danger of being blocked when no Sun Grid Engine service is available.

Qtcssh can operate in three different modes determining whether

- ❑ tasks are executed remotely.
- ❑ immediate or batch execution is requested.
- ❑ status output is verbose or only in case of any errors.

These modes either can be controlled by the command-line switches described below during *qtcsh* invocation or within an executing *qtcsh* via the built-in command *qrshmode* as described in section **BUILT-IN COMMANDS**.

OPTIONS

The options enlisted below are special to *qtcsh*. The user is referred to the *tcsh(1)* documentation for the explanation of further options.

-A

Switches *qtcsh* in verbose mode causing diagnostic output in case of remote execution.

-B

Switches remote task execution to batch mode. Tasks submitted to Sun Grid Engine will be queued if they cannot start immediately. As a consequence, *qtcsh* may block until the queued task can be started by Sun Grid Engine. While this behavior probably is undesirable during an interactive session, it may be very useful for execution of shell scripts through *qtcsh* as it avoids failure of the scripts due to temporarily unavailable resources for particular tasks.

-L

Switches off default behavior of remote execution of commands. Causes all commands to be executed locally even if they are contained in one of the *qtask(5)* files.

-R

Enforces remote execution of commands even if **JOB_ID** is set as environment variable.

BUILT-IN COMMANDS

This section only describes additional shell builtin commands which are not available in standard *tcsh(1)*.

qrshmode [-ANBILR]

Without options, the current operational mode of *qtcsh* is displayed. The options have the following effect:

- A switch to verbose output mode
- N switch to non-verbose output mode
- B switch to batch execution mode
- I switch to immediate execution mode
- L always execute commands locally
- R execute configured commands remotely

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of `/usr/CODINE` is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qtcsh* uses (in the order of precedence):

- The name of the cell specified in the environment variable `COD_CELL`, if it is set.

- The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qtcsh* client resides. Per default the local host is used.

FILES

`~/.qtask` user *qtask* file.
`<cod_root>/<cell>/common/qtask`
cluster *qtask* file.

SEE ALSO

sgc_intro(1), *qrsh(1)*, *qtask(5)* as well as *tcsh(1)* in `<cod_root>/3rd_party/qtcsh`.

COPYRIGHT

Qtsh contains portions of *tsh* which is copyrighted by The Regents of the University of California. Therefore, the following note applies: This product includes software developed by the University of California, Berkeley and its contributors.

See *sgc_intro(1)* and the information provided in `<cod_root>/3rd_party/qtsh` for a statement of further rights and permissions.

NAME

qsub – submit a batch job to Sun Grid Engine.
qsh – submit an interactive X-windows session to Sun Grid Engine.
qlogin – submit an interactive login session to Sun Grid Engine.
qrsh – submit an interactive rsh session to Sun Grid Engine.
qalter – modify a pending batch job of Sun Grid Engine.
qresub – submit a copy of an existing Sun Grid Engine job.

SYNTAX

qsub [options] [scriptfile | - [script_args]]
qsh [options] [-- xterm_args]
qlogin [options]
qrsh [options] [command [command_args]]
qalter [options] job/task_id_list [-- [script_args]]
qalter [options] -u user_list | -uall [-- [script_args]]
qresub [options] job_id_list

DESCRIPTION

Qsub submits batch jobs to the Sun Grid Engine queuing system. Sun Grid Engine supports single and multiple node jobs. **scriptfile** contains the commands to be run by the job using a shell (for example, *sh(1)* or *csh(1)*). Arguments to the job script are given by **script_args**. Sun Grid Engine flags may be entered as arguments to *qsub* or as embedded flags in the **scriptfile** if the first two characters of a script line either match '#\$' or are equal to the prefix string defined with the **-C** option described below.

Qsh submits an interactive X-windows session to Sun Grid Engine. An *xterm(1)* is brought up from the executing machine with the display directed either to the X-server indicated by the **DISPLAY** environment variable or as specified with the *-display qsh* option. Interactive jobs are not spooled if no resource is available to execute them. They are either dispatched to a suitable machine for execution immediately or the user submitting the job is notified by *qsh* that appropriate resources to execute the job are not available.

xterm_args are passed to the *xterm(1)* executable.

Qlogin is similar to *qsh* in that it submits an interactive job to the queueing system. It does not open an *xterm(1)* window on the X display, but uses the current terminal for user I/O. Usually, *qlogin* establishes a *telnet(1)* connection with the remote host, using standard client- and server-side commands. These commands can be configured with the **qlogin_daemon** (server-side, Sun Grid Engine *telnetd* if not set, otherwise something like */usr/sbin/in.telnetd*) and **qlogin_command** (client-side, Sun Grid Engine *telnet* if not set, otherwise something like */usr/bin/telnet*) parameters in the global and local configuration settings of *sge_conf(5)*. The client side command is automatically parameterized with the remote host name and port number to connect to (i.e. resulting in an invocation like */usr/bin/telnet my_exec_host 2442*). *Qlogin* is invoked exactly like *qsh* and its jobs can only run on INTERACTIVE queues. *Qlogin* jobs can only be used if the *cod_execd(8)* is running under the root account.

Qrsh is similar to *qlogin* in that it submits an interactive job to the queueing system. It uses the current terminal for user I/O. Usually, *qrsh* establishes a *rsh(1)* connection with the remote host. If no command is given to *qrsh*, a *rlogin(1)* session is established. The server-side commands used can be configured with the **rsh_daemon** and **rlogin_daemon** parameters in the global and local configuration settings of *sge_conf(5)*. A Sun Grid Engine *rshd* or *rlogind* is used, if the parameters are not set or otherwise something like */usr/sbin/in.rshd* or */usr/sbin/in.rlogind* needs to be configured. On the client-side, the **rsh_command** and **rlogin_command** parameters can be set in the global and local configuration settings of *sge_conf(5)*. If they are not set, *rsh(1)* and *rlogin(1)* binaries delivered with Sun Grid Engine are used. Use the cluster configuration to integrate mechanisms like *ssh* or the *rsh(1)* and *rlogin(1)* facilities supplied with the operating system.

Qrsh jobs can only run in INTERACTIVE queues unless the option **-now no** is used (see below). They can only be used, if the *cod_execd(8)* is running under the root account.

Qrsh provides an additional feature useful for the integration with interactive tools providing a specific command shell. If the environment variable **QSSH_WRAPPER** is set when *qrsh* is invoked, the command interpreter pointed to by **QSSH_WRAPPER** will be executed to run *qrsh* commands instead of the users login shell or any shell specified in the *qrsh* command-line.

Qalter can be used to change the attributes of pending jobs. Once a job is executing, changes are no longer possible. For job arrays, for which a part of the tasks can be pending and another part can be running (see the **-t** option below), modifications with *qalter* only affect the pending tasks. *Qalter* can change most of the characteristics of a job (see the corresponding statements in the OPTIONS section below), including those which were defined as embedded flags in the script file (see above).

Qresub allows to create jobs as copies from existing pending or running jobs. The copied jobs will have exactly the same attributes as the ones from which they are copied, but a new job ID. The only modification to the copied jobs supported by *qresub* is to assign a hold state with the **-h** option. This can be used to first copy a job and then change its attributes via *qalter*.

For *qsub*, *qsh*, *qrsh*, and *qlogin* the administrator and the user may define default request files (see *cod_request(5)*) which can contain any of the options described below. If an option in a default request file is understood by *qsub* and *qlogin* but not by *qsh* the option is silently ignored if *qsh* is invoked. Thus you can maintain shared default request files for both *qsub* and *qsh*.

A cluster wide default request file may be placed under

`$CODINE_ROOT/$COD_CELL/common/cod_request`. User private default request files are processed under the locations `$HOME/.cod_request` and `$cwd/.cod_request`. The working directory local default request file has the highest precedence, then the home directory located file and then the cluster global file. The option arguments, the embedded script flags and the options in the default request files are processed in the following order:

- left to right in the script line,
- left to right in the default request files,
- from top to bottom of the script file (*qsub* only),
- from top to bottom of default request files,
- from left to right of the command line.

In other words, the command line can be used to override the embedded flags and the default request settings. The embedded flags, however, will override the default settings.

Note – The *-clear* option can be used to discard any previous settings at any time in a default request file, in the embedded script flags, or in a command-line option. It is, however, not available with *qalter*.

The options described below can be requested either hard or soft. By default, all requests are considered hard until the **-soft** option (see below) is encountered. The hard/soft status remains in effect until its counterpart is encountered again. If all the hard requests for a job cannot be met, the job will not be scheduled. Jobs which cannot be run at the present time remain spooled.

OPTIONS

-@ optionfile

Forces *qsub*, *qrsh*, *qsh*, or *qlogin* to use the options contained in **optionfile**. The indicated file may contain all valid options. Comment lines are starting with a “#” sign.

-a date_time

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines the time and date at which a job is eligible for execution. **Date_time** conforms to `[[CC]]YY]MMDDhhmm.[ss]`, where:

- CC** denotes the century in 2 digits.
- YY** denotes the year in 2 digits.
- MM** denotes the month in 2 digits.
- DD** denotes the day in 2 digits.
- hh** denotes the hour in 2 digits.

mm denotes the minute in 2 digits.

ss denotes the seconds in 2 digits (default 00).

If any of the optional date fields is omitted, the corresponding value of the current date is assumed.

Usage of this option may cause unexpected results if the clocks of the hosts in the Sun Grid Engine pool are out of sync. Also, the proper behavior of this option very much depends on the correct setting of the appropriate timezone, e.g. in the TZ environment variable (see *date(1)* for details), when the Sun Grid Engine daemons *cod_qmaster(8)* and *cod_execd(8)* are invoked.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-ac variable[=value],...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Adds the given name/value pair(s) to the job's context. **Value** may be omitted. Sun Grid Engine appends the given argument to the list of context variables for the job. Multiple **-ac**, **-dc**, and **-sc** options may be given. The order is important here.

Qalter allows changing this option even while the job executes.

-A account_string

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Identifies the account to which the resource consumption of the job should be charged. The **account_string** may be any arbitrary ASCII alphanumeric string but may contain no blank or separator characters. The underbar '_' is considered a non-separator. In the absence of this parameter Sun Grid Engine will place the default account string "cod" in the accounting record of the job.

Qalter allows changing this option even while the job executes.

-c occasion_specifier

Available for *qsub* and *qalter* only.

Defines or redefines whether the job should be checkpointed, and if so, under what circumstances. The specification of the checkpointing occasions with this option overwrites the definitions of the *when* parameter in the checkpointing environment (see *checkpoint(5)*) referenced by the *qsub -ckpt* switch. Possible values for **occasion_specifier** are

- n no checkpoint is performed.
- s checkpoint when batch server is shut down.
- m checkpoint at minimum CPU interval.
- x checkpoint when job gets suspended.
- <interval> checkpoint in the specified time interval.

The minimum CPU interval is defined in the queue configuration (see *queue_conf(5)* for details).

<interval> has to be specified in the format hh:mm:ss. The maximum of <interval> and the queue's minimum CPU interval is used if <interval> is specified. This is done to ensure that a machine is not overloaded by checkpoints being generated too frequently.

-ckpt ckpt_name

Available for *qsub* and *qalter* only.

Selects the checkpointing environment (see *checkpoint(5)*) to be used for a checkpointing the job. Also declares the job to be a checkpointing job.

-clear

Available for *qsub*, *qrsh*, *qsh*, and *qlogin* only.

Causes all elements of the job to be reset to the initial default status prior to applying any modifications (if any) appearing in this specific command.

-cwd

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Execute the job from the current working directory. This switch will activate Sun Grid Engine's path aliasing facility, if the corresponding configuration files are present (see *codine_aliases(5)*).

In case of *qalter*, the previous definition of the current working directory will be overwritten, if *qalter* is executed from a different directory than the preceding *qsub* or *qalter*.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-C prefix_string

Available for *qsub* only.

Prefix_string defines the prefix that declares a directive to *qsub* in the job's scriptfile. The prefix is not a job attribute, but affects the behavior of *qsub*. If the -C option is presented with the value of the directive prefix as a null string, *qsub* will not scan the scriptfile

The directive prefix consists of two ASCII characters which when appearing in the first two bytes of a script line indicate that what follows is a Sun Grid Engine command (default is "#\$").

The user should be aware that changing the first delimiter character can produce unforeseen side effects. If the script file contains anything other than a "#" character in the first byte position of the line, the shell processor for the job will reject the line and may exit the job prematurely.

If the -C option is present in the script file, it is ignored.

-dc variable,...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Removes the given variable(s) from the job's context. Multiple **-ac**, **-dc**, and **-sc** options may be given. The order is important here.

Qalter allows changing this option even while the job executes.

-display display_specifier

Available for *qsh* only.

Directs *xterm(1)* to use **display_specifier** in order to contact the X server.

-dl date_time

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only. This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Specifies the deadline initiation time in [[CC]YY]DDhhmm[.SS] format (see **-a** option above). The deadline initiation time is the time at which a deadline job has to reach top priority to be able to complete within a given deadline. Before the deadline initiation time the priority of a deadline job will be raised steadily until it reaches the maximum as configured by the Sun Grid Engine administrator.

This option is applicable for users allowed to submit deadline jobs only.

-e [hostname:]path,...

Available for *qsub* and *qalter* only.

Defines or redefines the path used for the standard error stream of the job. If the **path** constitutes an absolute path name, the error-path attribute of the job is set to its value including the **hostname**. If the path name is relative, Sun Grid Engine expands **path** either with the current working directory path in case the **-cwd** (see above) switch is also specified or with the home directory path otherwise. If **hostname** is present, the standard error stream will be placed under the corresponding location if the job runs on the specified host.

By default the file name for standard error has the form *job_name.ejob_id* and *job_name.ejob_id.task_id* for job array tasks (see **-t** option below).

If **path** is a directory, the standard error stream of the job will be put in this directory under the default file name. If the pathname contains certain pseudo environment variables, their value will be expanded at runtime of the job and will be used to constitute the standard error stream path name. The following pseudo environment variables are supported currently:

| | |
|----------------------|---|
| \$HOME | home directory on execution machine |
| \$USER | user ID of job owner |
| \$JOB_ID | current job ID |
| \$JOB_NAME | current job name (see -N option) |
| \$HOSTNAME | name of the execution host |
| \$COD_TASK_ID | job array task index number |

Alternatively to **\$HOME** the tilde sign “~” can be used as common in *csh(1)* or *ksh(1)*.

Note – The “~” sign also works in combination with user names, so that “~<user>” expands to the home directory of <user>. Using another user ID than that of the job owner requires corresponding permissions, of course.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-hard

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Signifies that all resource requirements following in the command line will be hard requirements and must be satisfied in full before a job can be scheduled.

As Sun Grid Engine scans the command line and script file for Sun Grid Engine options and parameters it builds a list of resources required by a job. All such resource requests are considered as absolutely essential for the job to commence. If the **-soft** option (see below) is encountered during the scan then all following resources are designated as “soft requirements” for execution, or “nice-to-have, but not essential”. If the **-hard** flag is encountered at a later stage of the scan, all resource requests following it once again become “essential”. The **-hard** and **-soft** options in effect act as “toggles” during the scan.

-h | -H {u|s|o|n|U|S|O|N}...

Available for *qsub*, *qrsh*, *qsh*, *qlogin*, *qalter* and *qresub*.

List of holds to place on the job.

- ‘u’ denotes a user hold.
- ‘s’ denotes a system hold.
- ‘o’ denotes a operator hold.
- ‘n’ denotes no hold.

As long as any hold other than ‘n’ is assigned to the job the job is not eligible for execution. Holds can be released via *qalter* and *qrls(1)*. In case of *qalter* this is supported by the following additional option specifiers for the **-h** switch:

- ‘U’ removes a user hold.
- ‘S’ removes a system hold.
- ‘O’ removes a operator hold.

Sun Grid Engine managers can assign and remove all hold types, Sun Grid Engine operators can assign and remove user and operator holds and users can only assign or remove user holds.

In the case of *qsub* only user holds can be placed on a job and thus only the first form of the option with the **-h** switch alone is allowed. As opposed to this, *qalter* requires the second form described above.

An alternate means to assign hold is provided by the *qhold(1)* facility.

If the job is a job array (see the **-t** option below), all tasks specified via **-t** are affected by the **-h** operation simultaneously.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-help

Prints a listing of all options.

-hold_jid job_id,...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines the job dependency list of the submitted job. The submitted job is not eligible for execution unless all jobs referenced in the comma separated job id list have completed successfully.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-inherit

Available only for *qrsh* and *qmake(1)*.

qrsh allows to start a task in an already scheduled parallel job. The option **-inherit** tells *qrsh* to read a job id from the environment variable **JOB_ID** and start the specified command as a task in this job. Please note that in this case, the hostname of the host where the command shall be executed, must precede the command to execute; the syntax changes to

**qrsh -inherit [other options] hostname
command [command_args]**

Note also, that in combination with **-inherit**, most other command line options will be ignored. Only the options **-verbose**, **-v** and **-V** will be interpreted. As a replacement to option **-cwd** please use **-v PWD**.

Usually a task should have the same environment (including the current working directory) as the corresponding job, so specifying the option **-V** should be suitable for most applications.

Note – If in your system the *commd* port is not configured as service, but via environment variable **COMMD_PORT**, make sure that this variable is set in the environment when calling *qrsh* or *qmake* with option **-inherit**. If you call *qrsh* or *qmake* with option **-inherit** from within a job script, export **COMMD_PORT** with the submit option or special comment **"-v COMMD_PORT"**.

-j yln

Available for *qsub* and *qalter* only.

Specifies whether or not the standard error stream of the job is merged into the standard output stream.

If both the **-j y** and the **-e** options are present, Sun Grid Engine sets, but ignores the error-path attribute.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-l resource=value,...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Launch the job in a Sun Grid Engine queue meeting the given resource request list. In case of *qalter* the previous definition is replaced by the specified one.

complex(5) describes how a list of available resources and their associated valid value specifiers can be obtained.

There may be multiple **-l** switches in a single command. You may request multiple **-l** options to be soft or hard both in the same command line. In case of a serial job multiple **-l** switches refine the definition for the sought queue.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-m blalalnl,...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines under which circumstances mail is to be sent to the job owner or to the users defined with the **-M** option described below. The option arguments have the following meaning:

‘b’ Mail is sent at the beginning of the job.

‘e’ Mail is sent at the end of the job.

‘a’ Mail is sent when the job is aborted.

‘s’ Mail is sent when the job is suspended.

‘n’ No mail is sent.

Currently no mail is sent when a job is suspended.

For *qsh* and *qlogin* mail at the beginning or end of the job is suppressed when it is encountered in a default request file.

Qalter allows changing the b, e, and a option arguments even while the job executes. The modification of the b option argument will only be in effect after a restart or migration of the job, however.

-M user[@host],...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines the list of users to which the server that executes the job has to send mail, if the server sends mail about the job. Default is the job owner at the originating host.

Qalter allows changing this option even while the job executes.

-masterq queue,...

Available for *qsub*, *qsh*, *qrsh*, *qlogin* and *qalter*. Only meaningful for parallel jobs, i.e. together with the **-pe** option.

Defines or redefines a list of queues which may be used to become the so called *master queue* of this parallel job. The *master queue* is defined as the queue where the parallel job is started. The other queues to which the parallel job spawns tasks are called *slave queues*. A parallel job only has one *master queue*.

This parameter has all the properties of a resource request and will be merged with requirements derived from the **-l** option described above.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-notify

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

This flag, when set causes Sun Grid Engine to send “warning” signals to a running job prior to sending the signals themselves. If a SIGSTOP is pending the job will receive a SIGUSR1 several seconds before the SIGSTOP. If a SIGKILL is pending the job will receive a SIGUSR2 several seconds before the SIGKILL. The amount of time delay is controlled by the **notify** parameter in each queue configuration (see *queue_conf(5)*).

Note – The Linux operating system “misuses” the user signals SIGUSR1 and SIGUSR2 in its current Posix thread implementation. You might not want to use the **-notify** option if you are running threaded applications in your jobs under Linux.

Qalter allows changing this option even while the job executes.

-now y[es]n[o]

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

-now y tries to start the job immediately or not at all. The command returns 0 on success, or 1 on failure (also if the job could not be scheduled immediately). **-now y** is default for *qsh*, *qlogin* and *qrsh*

With option **-now n** the job will be put into the pending queue, if it cannot be executed immediately.

-now n is default for *qsub*.

-N name

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

The name of the job. The name can be any printable set of characters.

If the **-N** option is not present Sun Grid Engine assigns the name of the job script to the job after any directory pathname has been removed from the script-name. If the script is read from standard input the job name defaults to STDIN.

In case of *qsh* or *qlogin* and if the **-N** option is absent the string ‘INTERACT’ is assigned to the job.

Qalter allows changing this option even while the job executes.

-nostdin

Available only for *qrsh*.

Suppress the input stream STDIN - *qrsh* will pass the option -n to the *rsh(1)* command. This is especially usefull, if multiple tasks are executed in parallel using *qrsh*, e.g. in a *make(1)* process - it would be undefined, which process would get the input.

-o [hostname:]path,...

Available for *qsub* and *qalter* only.

The path used for the standard output stream of the job. The **path** is handled as described in the **-e** option for the standard error stream.

By default the file name for standard output has the form *job_name.ojob_id* and

job_name.job_id.task_id for job array tasks (see **-t** option below).

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-ot override_tickets

Available for *qalter* only. This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Changes the number of override tickets for the specified job. Requires manager/operator privileges.

-P project_name

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only. This option is only supported in case of a Sun Grid Engine, Enterprise Edition system. It is not available for Sun Grid Engine systems.

Specifies the project to which this job is assigned. The administrator needs to give permission to individual users to submit jobs to a specific project. (see **-apri** option to *qconf(1)*).

-p priority

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines the priority of the job relative to other jobs. Priority is an integer in the range -1023 to 1024. The default priority value for the jobs is 0.

In a Sun Grid Engine system, users may only decrease the priority of their jobs. Sun Grid Engine managers and administrators may also increase the priority associated with jobs. If a pending job has higher priority, it is earlier eligible for being dispatched by the Sun Grid Engine scheduler. The job priority has no effect on running jobs in Sun Grid Engine.

In Sun Grid Engine, Enterprise Edition, the job priority influences the Share Tree Policy and the Functional Policy. It has no effect on the **Deadline** and **Override** Policies (see *share_tree(5)*, *sched_conf(5)* and the *Sun Grid Engine, Enterprise Edition Installation and Administration Guide* for further information on the resource management policies supported by Sun Grid Engine, Enterprise Edition).

In case of the Share Tree Policy, users can distribute the tickets, to which they are currently entitled, among their jobs using different priorities assigned via **-p**. If all jobs have the same priority value, the tickets are distributed evenly. Jobs receive tickets relative to the different priorities otherwise. Priorities are treated like an additional level in the share tree in the latter case.

In connection with the Functional Policy, the priority can be used to weight jobs within the functional job category. Again tickets are distributed relative to any uneven priority distribution treated as a virtual share distribution level underneath the functional job category.

If both, the Share Tree and the Functional Policy are active, the job priorities will have an effect in both policies and the tickets independently derived in each of them are added up to the total number of tickets for each job.

-pe parallel_environment n[-[m]][-]m,...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Parallel programming environment (PE) to instantiate. The range descriptor behind the PE name specifies the number of parallel processes to be run. Sun Grid Engine will allocate the appropriate resources as

available. The *sge_pe(5)* manual page contains information about the definition of PEs and about how to obtain a list of currently valid PEs.

You can specify the PE name by using the wildcard character “*”, thus the request “pvm*” will match any parallel environment with a name starting with the string “pvm”.

The range specification is a list of range expressions of the form *n-m* (*n* as well as *m* being positive non-zero integer numbers), where *m* is an abbreviation for *m-m*, *-m* is a short form for *1-m* and *n-* is an abbreviation for *n-infinity*. The range specification is processed as follows: The largest number of queues requested is checked first. If enough queues meeting the specified attribute list are available, all are allocated. The next smaller number of queues is checked next and so forth.

If additional **-l** options are present, they restrict the set of eligible queues for the parallel job.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-q queue,...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines a list of queues which may be used to execute this job. This parameter has all the properties of a resource request and will be merged with requirements derived from the **-l** option described above.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-qs_args ... -qs_end

Available for *qsub* and *qalter* only.

Valid with the Queuing System Interface (QSI) option only. Please ask your system administrator.

The options between **-qs_args** and **-qs_end** braces are passed from Sun Grid Engine to a foreign queuing system which interfaced via the Sun Grid Engine QSI facility.

-r yln

Available for *qsub* and *qalter* only.

Identifies the ability of a job to be rerun or not. If the value of **-r** is ‘y’, rerun the job if the job was aborted without leaving a consistent exit state (this is typically the case if the node on which the job is running crashes). If **-r** is ‘n’, do not rerun the job under any circumstances.

Interactive jobs submitted with *qsh* or *qlogin* are not re-runnable.

Qalter allows changing this option even while the job executes.

-t n[-m[:s]]

Available for *qsub* and *qalter* only.

Submits a so called *Job Array*, i.e. an array of identical tasks being only differentiated by an index number and being treated by Sun Grid Engine almost like a series of jobs. The option argument to **-t** specifies the number of job array tasks and the index number which will be associated with the tasks. The index numbers will be exported to the job tasks via the environment variable **COD_TASK_ID**.

The task id range specified in the option argument may be a single number, a simple range of the form n-m or a range with a step size. Hence, the task id range specified by 2-10:2 would result in the task id indexes 2, 4, 6, 8, and 10, i.e. in a total of 5 tasks identical tasks with the environment variable **COD_TASK_ID** containing one of the 5 index numbers each.

All job array tasks inherit the same resource requests and attribute definitions as specified in the *qsub* or *qalter* command line, except for the **-t** option. The tasks are scheduled independently and, provided enough resources, concurrently very much like separate jobs. However, a job array or a sub-array thereof can be accessed as a total by commands like *qmod(1)* or *qdel(1)*. See the corresponding manual pages for further detail.

Job arrays are commonly used to execute the same type of operation on varying input data sets correlated with the task index number. The number of tasks in a job array is unlimited.

STDOUT and STDERR of job array tasks will be written into different files with the default location `<jobname>.[‘e’|‘o’]<job_id>’.‘<task_id>`

In order to change this default, the **-e** and **-o** options (see above) can be used together with the pseudo environment variables \$HOME, \$USER, \$JOB_ID, \$JOB_NAME, \$HOSTNAME, and \$COD_TASK_ID.

Note – You can use the output redirection to divert the output of all tasks into the same file, but the result of this is undefined.

-sc variable[=value],...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Sets the given name/value pairs as the job’s context. **Value** may be omitted. Sun Grid Engine replaces the job’s previously defined context with the one given as the argument. Multiple **-ac**, **-dc**, and **-sc** options may be given. The order is important here.

Contexts are a way to dynamically attach and remove meta-information to and from a job. The context variables are **not** passed to the job’s execution context in its environment.

Qalter allows changing this option even while the job executes.

-soft

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Signifies that all resource requirements following in the command line will be soft requirements and are to be filled on an “as available” basis.

As Sun Grid Engine scans the command line and script file for Sun Grid Engine options and parameters it builds a list of resources required by a job. All such resource requests are considered as absolutely essential for the job to commence. If the **-soft** option is encountered during the scan then all following resources are designated as “soft requirements” for execution, or “nice-to-have, but not essential”. If the **-hard** flag (see above) is encountered at a later stage of the scan, all resource requests following it once again become “essential”. The **-hard** and **-soft** options in effect act as “toggles” during the scan.

-S [host:]pathname,...

Available for *qsub*, *qsh*, *qlogin* and *qalter*.

Specifies the interpreting shell for the job. Only one **pathname** component without a **host** specifier is valid and only one path name for a given host is allowed. Shell paths with host assignments define the interpreting shell for the job if the host is the execution host. The shell path without host specification is used if the execution host matches none of the hosts in the list.

Furthermore, the pathname can be constructed with pseudo environment variables as described for the **-e** option above.

In the case of *qsh* the specified shell path is used to execute the corresponding command interpreter in the *xterm(1)* (via its **-e** option) started on behalf of the interactive job.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-u username,... | -uall

Available for *qalter* only. Changes are only made on those jobs which were submitted by users specified in the list of usernames. For managers it is possible to use the **qalter -uall** command to modify all jobs of all users.

If you use the **-u** or **-uall** switch it is not permitted to specify an additional *job/task_id_list*.

-v variable[=value],...

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Defines or redefines the environment variables to be exported to the execution context of the job. If the **-v** option is present Sun Grid Engine will add the environment variables defined as arguments to the switch and, optionally, values of specified variables, to the execution context of the job.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

-verbose

Available only for *qrsh* and *qmake(1)*.

Unlike *qsh* and *qlogin*, *qrsh* does not output any informational messages while establishing the session compliant with the standard *rsh(1)* and *rlogin(1)* system calls. If the option **-verbose** is set, *qrsh* behaves as verbose as the *qsh* and *qlogin* commands and outputs informations about the process of establishing the *rsh(1)* or *rlogin(1)* session.

-verify

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Does not submit a job but prints information on the job as being represented by the current command-line and all pertinent external influences.

-V

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Specifies that all environment variables active within the *qsub* utility be exported to the context of the job.

-w elwnlv

Available for *qsub*, *qrsh*, *qsh*, *qlogin* and *qalter* only.

Specifies a validation level applied to the job to be submitted (*qsub*, *qlogin*, and *qsh*) or the specified queued job (*qalter*). The information displayed indicates whether the job possibly can be scheduled assuming an empty system with no other jobs. Resource requests exceeding the configured maximal thresholds or requesting unavailable resource attributes are possible causes for jobs to fail this validation.

The specifiers e, w, n and v define the following validation modes:

- ‘e’ error - jobs with invalid requests will be rejected; the default for *qrsh*, *qsh* and *qlogin*.
- ‘w’ warning - only a warning will be displayed for invalid requests.
- ‘n’ none - switches off validation; the default for *qalter* and *qsub*.
- ‘v’ verify - does not submit the job but prints extensive validation report.

Note – The necessary checks are performance consuming and hence the checking is switched off by default.

Note – The reasons for job requirements being invalid with respect to resource availability of queues are displayed in the “-w v” case using the format as described for the *qstat(1)* -F option (see description of Full Format in section OUTPUT FORMATS of the *qstat(1)* manual page).

job/task_id_list

Specified by the following form:

job_id[.task_range][.job_id[.task_range],...]

If present, the *task_range* restricts the effect of the operation to the job array task range specified as suffix to the job id (see the **-t** option to *qsub(1)* for further details on job arrays).

The task range specifier has the form n[-m[:s]]. The range may be a single number, a simple range of the form n-m or a range with a step size.

Instead of job/task_id_list it is possible to use the keyword ‘all’ to modify all jobs of the current user.

scriptfile

Available for *qsub* only.

The job’s scriptfile. If not present or if the operand is the single-character string ‘-’, *qsub* reads the script from standard input.

script_args

Available for *qsub* and *qalter* only.

Arguments to the job. Not valid if the script is entered from standard input.

Qalter allows changing this option even while the job executes. The modified parameter will only be in effect after a restart or migration of the job, however.

xterm_args

Available for *qsh* only.

Arguments to the *xterm(1)* executable, as defined in the configuration. For details, refer to *sge_conf(5)*.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *qsub*, *qsh*, *qlogin* or *qalter* use (in the order of precedence):

The name of the cell specified in the environment

variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *qsub*, *qsh*, *qlogin* or *qalter* client resides. Per default the local host is used.

In addition to those environment variables specified to be exported to the job via the **-v** or the **-V** option (see above) *qsub*, *qsh*, and *qlogin* add the following variables with the indicated values to the variable list:

COD_O_HOME

the home directory of the submitting client.

COD_O_HOST

the name of the host on which the submitting client is running.

COD_O_LOGNAME

the LOGNAME of the submitting client.

COD_O_MAIL

the MAIL of the submitting client. This is the mail directory of the submitting client.

COD_O_PATH

the executable search path of the submitting client.

COD_O_SHELL

the SHELL of the submitting client.

COD_O_TZ

the time zone of the submitting client.

COD_O_WORKDIR

the absolute path of the current working directory of the submitting client.

Furthermore, Sun Grid Engine sets additional variables into the job's environment, as listed below.

Note – Several variables (as denoted below) are also set for jobs forwarded to another queuing system via the Sun Grid Engine Queuing System Interface (the QSI needs to be licensed and installed as an add-on product – ask your system administrator).

ARC

The Sun Grid Engine architecture name of the node on which the job is running. The name is compiled-in into the *cod_execd(8)* binary.

COD_CKPT_ENV

Specifies the checkpointing environment (as selected with the **-ckpt** option) under which a checkpointing job executes. Only set for checkpointing jobs.

COD_CKPT_DIR

Only set for checkpointing jobs. Contains path *ckpt_dir* (see *checkpoint(5)*) of the checkpoint interface.

COD_STDERR_PATH

the pathname of the file to which the standard error stream of the job is diverted. Commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start/stop or checkpointing scripts.

COD_STDOUT_PATH

the pathname of the file to which the standard output stream of the job is diverted. Commonly used for enhancing the output with messages from prolog, epilog, parallel environment start/stop or checkpointing scripts.

COD_TASK_ID

The index number of the current job array task (see **-t** option above). This is an unique number in each job array and can be used to reference different input data records, for example. This environment variable is not set for non-array jobs.

COD_JOB_SPOOL_DIR

The directory used by *cod_shepherd(8)* to store job related data during job execution. This directory is owned by root or by a Sun Grid Engine administrative account and commonly is not open for read or write access to regular users.

ENVIRONMENT

The **ENVIRONMENT** variable is set to **BATCH** to identify that the job is being executed under Sun Grid Engine control. Also set for QSI jobs.

HOME

The user's home directory path from the *passwd(5)* file.

HOSTNAME

The hostname of the node on which the job is running.

JOB_ID

A unique identifier assigned by the *cod_qmaster(8)* when the job was submitted. The job **ID** is a decimal integer in the range 1 to 99999. Also set for QSI jobs.

JOB_NAME

The job name, either 'INTERACT' for interactive jobs or built from the *qsub* script filename, a period, and the digits of the job **ID**. This default may be overwritten by the **-N** option. Also set for QSI jobs.

LAST_HOST

The name of the preceding host in case of migration of a checkpointing job. Also set for QSI jobs.

LOGNAME

The user's login name from the *passwd(5)* file.

NHOSTS

The number of hosts in use by a parallel job.

NQUEUES

The number of queues allocated for the job (always 1 for serial jobs). Also set for QSI jobs.

NSLOTS

The number of queue slots in use by a parallel job.

PATH

A default shell search path of:

/usr/local/bin:/usr/ucb:/bin:/usr/bin

PE

The parallel environment under which the job executes (for parallel jobs only).

PE_HOSTFILE

The path of a file containing the definition of the virtual parallel machine assigned to a parallel job by Sun Grid Engine. See the description of the **\$pe_hostfile** parameter in *sge_pe(5)* for details on the format of this file. The environment variable is only available for parallel jobs.

QUEUE

The name of the queue in which the job is running. Also set for QSI jobs.

REQUEST

Available for batch jobs only.

The request name of a job as specified with the **-N** switch (see above) or taken as the name of the job script file. Also set for QSI jobs.

RESTARTED

This variable is set to 1 if a job was restarted either after a system crash or after a migration in case of a checkpointing job. The variable has the value 0 otherwise.

SHELL

The user's login shell from the *passwd(5)* file.

Note – This is not necessarily the shell in use for the job.

TMPDIR

The absolute path to the job's temporary working directory.

TMP

The same as **TMPDIR**; provided for compatibility with NQS.

TZ

The time zone variable imported from *cod_execd(8)* if set.

USER

The user's login name from the *passwd(5)* file.

RESTRICTIONS

There is no controlling terminal for batch jobs under Sun Grid Engine and any tests or actions on a controlling terminal will fail. If these operations are in your *.login* or *.cshrc* file, they will possibly cause your job to abort.

Insert the following test before any commands that are not pertinent to batch jobs in your *.login*:

```
if ( $?JOB_NAME) then
    echo "Sun Grid Engine spooled job"
    exit 0
endif
```

Don't forget to set your shell's search path in your shell start-up before this code.

EXAMPLES

The following is the simplest form of a Sun Grid Engine script file.

```
#!/bin/csh
a.out
```

The next example is a more complex Sun Grid Engine script.

```
#!/bin/csh
# Force csh
#$ -S /bin/csh

# Which account to be charged cpu time
#$ -A santa_claus

# date-time to run, format [[CC]yy]MMDDhhmm[.SS]
#$ -a 12241200

# to run I want 6 or more parallel processes
# under the PE pvm. the processes require
# 128M of memory
#$ -pe pvm 6- -l mem=128
# If I run on dec_x put stderr in /tmp/foo, if I
# run on sun_y, put stderr in /usr/me/foo
#$ -e dec_x:/tmp/foo,sun_y:/usr/me/foo

# Send mail to these users
#$ -M santa@heaven,claus@heaven
# Mail at beginning/end/on suspension
#$ -m bes

# Export these environmental variables
#$ -v PVM_ROOT,FOOBAR=BAR

# The job is located in the current
# working directory.
#$ -cwd

a.out
```

FILES

| | |
|--|--------------------------------|
| <code>\$REQUEST.oJID[.TASKID]</code> | STDOUT of job #JID |
| <code>\$REQUEST.eJID[.TASKID]</code> | STDERR of job |
| <code>\$REQUEST.poJID[.TASKID]</code> | STDOUT of par. env. of job |
| <code>\$REQUEST.peJID[.TASKID]</code> | STDERR of par. env. of job |
| <code>\$REQUEST.hostsJID[.TASKID]</code> | hosts file of par. env. of job |
| <code>\$cwd/.codine_aliases</code> | cwd path aliases |
| <code>\$cwd/.cod_request</code> | cwd default request |

| | |
|---|----------------------------------|
| <code>\$HOME/.codine_aliases</code> | user path aliases |
| <code>\$HOME/.cod_request</code> | user default request |
| <code><cod_root>/<cell>/common/.codine_aliases</code> | |
| | cluster path aliases |
| <code><cod_root>/<cell>/common/.cod_request</code> | |
| | cluster default request |
| <code><cod_root>/<cell>/common/act_qmaster</code> | |
| | Sun Grid Engine master host file |

SEE ALSO

sgc_intro(1), *qconf(1)*, *qdel(1)*, *qhold(1)*, *qmod(1)*, *qrls(1)*, *qstat(1)*, *accounting(5)*, *cod_aliases(5)*, *sgc_conf(5)*, *cod_request(5)*, *sgc_pe(5)*, *complex(5)*.

COPYRIGHT

If configured correspondingly, *qrsh* and *qlogin* contain portions of the *rsh*, *rshd*, *telnet* and *telnetd* code copyrighted by The Regents of the University of California. Therefore, the following note applies with respect to *qrsh* and *qlogin*: This product includes software developed by the University of California, Berkeley and its contributors.

See *sgc_intro(1)* as well as the information provided in `<cod_root>/3rd_party/qrsh` and `<cod_root>/3rd_party/qlogin` for a statement of further rights and permissions.

ACCESS_LIST(5)

NAME

`access_list` – Sun Grid Engine access list file format

DESCRIPTION

Access lists are used in Sun Grid Engine to define access permissions of users to queues (see *queue_conf(5)*) or parallel environments (see *sge_pe(5)*). A list of currently configured access lists can be displayed via the *qconf(1)* **-su** option. The contents of each enlisted access list can shown via the **-su** switch. The output follows the *access_list* format description. New access lists can be created and existing can be modified via the **-au** and **-du** options to *qconf(1)*.

FORMAT

Each user or UNIX user group appears in a single lines. Only symbolic names are allowed. A group is differentiated from a user name by prefixing the group name with a '@' sign.

SEE ALSO

sge_intro(1), *qconf(1)*, *sge_pe(5)*, *queue_conf(5)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

accounting – Sun Grid Engine accounting file format

DESCRIPTION

An accounting record is written to the Sun Grid Engine accounting file for each job having finished. The accounting file is processed by *qacct(1)* to derive accounting statistics.

FORMAT

Each job is represented by a line in the accounting file. Empty lines and lines which contain one character or less are ignored. Accounting record entries are separated by colon (':') signs. The entries denote in their order of appearance:

qname

Name of the queue in which the job has run.

hostname

Name of the execution host.

group

The effective group id of the job owner when executing the job.

owner

Owner of the Sun Grid Engine job.

job_name

Job name.

job_number

Job identifier - job number.

account

An account string as specified by the *qsub(1)* or *qalter(1)* **-A** option.

priority

Priority value assigned to the job corresponding to the **priority** parameter in the queue configuration (see *queue_conf(5)*).

submission_time

Submission time in seconds (since epoch format).

start_time

Start time in seconds (since epoch format).

end_time

End time in seconds (since epoch format).

failed

Indicates the problem which occurred in case a job could not be started on the execution host (e.g. because the owner of the job did not have a valid account on that machine). If Sun Grid Engine tries to start a job multiple times, this may lead to multiple entries in the accounting file corresponding to the same job **ID**.

exit_status

Exit status of the job script (or Sun Grid Engine specific status in case of certain error conditions).

ru_wallclock

Difference between **end_time** and **start_time** (see above).

The remainder of the accounting entries follows the contents of the standard UNIX *rusage* structure as described in *getrusage(2)*. The following entries are provided:

ru_utime

ru_stime

ru_maxrss

ru_ixrss

ru_ismrss

ru_idrss

ru_isrss

ru_minflt

ru_majflt

ru_nswap

ru_inblock

ru_oublock

ru_msgsnd

ru_msgrcv

ru_nsignals

ru_nvcsw

ru_nivcsw

project

The project which was assigned to the job. Projects are only supported in case of a Sun Grid Engine, Enterprise Edition system.

department

The department which was assigned to the job. Departments are only supported in case of a Sun Grid Engine, Enterprise Edition system.

granted_pe

The parallel environment which was selected for that job.

slots

The number of slots which were dispatched to the job by the scheduler.

task_number

Job array task index number.

cpu

The cpu time usage in seconds. Only supported in case of a Sun Grid Engine, Enterprise Edition system.

mem

The integral memory usage in Gbytes seconds. Only supported in case of a Sun Grid Engine, Enterprise Edition system.

io

The amount of data transferred in input/output operations. Only supported in case of a Sun Grid Engine, Enterprise Edition system.

SEE ALSO

sge_intro(1), *qacct(1)*, *qalter(1)*, *qsub(1)*, *getrusage(2)*, *queue_conf(5)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

CALENDAR_CONF(5)

NAME

calendar_conf – Sun Grid Engine calendar configuration file format

DESCRIPTION

calendar_conf reflects the format of the Sun Grid Engine calendar configuration. The definition of calendars is used to specify “on duty” and “off duty” time periods for Sun Grid Engine queues on a time of day, day of week or day of year basis. Various calendars can be implemented and the appropriate calendar definition for a certain class of jobs can be attached to a queue.

calendar_conf entries can be added, modified and displayed with the *-Acal*, *-acal*, *-Mcal*, *-mcal*, *-scal* and *-scall* options to *qconf(1)* or with the calendar configuration dialog of the graphical user interface *qmon(1)*. The format of the calendar configuration entries is defined as follows:

FORMAT

calendar_name

The name of the calendar to be used when attaching it to queues or when administering the calendar definition.

year

The queue status definition on a day of the year basis. This field generally will specify on which days of a year (and optionally at which times on those days) a queue, to which the calendar is attached, will change to a certain state. The syntax of the **year** field is defined as follows:

```
year:=  
    {year_day_range_list[=daytime_range_list][=state]  
    | [year_day_range_list=]daytime_range_list[=state]  
    | [year_day_range_list=][daytime_range_list=]state} ...
```

Where

- at least one of **year_day_range_list**, **daytime_range_list** and **state** always have to be present,

- every day in the year is assumed if **year_day_range_list** is omitted,
- all day long is assumed if **daytime_range_list** is omitted,
- switching the queue to “off” (i.e. disabling it) is assumed if **state** is omitted,
- the queue is assumed to be enabled for days neither referenced implicitly (by omitting the **year_day_range_list**) nor explicitly

and the syntactical components are defined as follows:

```

year_day_range_list := {yearday-yearday | yearday},...
daytime_range_list := hour[:minute][:second]-
                        hour[:minute][:second],...
state := {on | off | suspended}
year_day := month_day.month.year
month_day := {1 | 2 | ... | 31}
month := {jan | feb | ... | dec | 1 | 2 | ... | 12}
year := {1970 | 1971 | ... | 2037}

```

week

The queue status definition on a day of the week basis. This field generally will specify on which days of a week (and optionally at which times on those days) a queue, to which the calendar is attached, will change to a certain state. The syntax of the **week** field is defined as follows:

```

week :=
    {week_day_range_list[=daytime_range_list][=state]
    | [week_day_range_list=]daytime_range_list[=state]
    | [week_day_range_list][daytime_range_list=]state} ...

```

Where

- at least one of **week_day_range_list**, **daytime_range_list** and **state** always have to be present,
- every day in the week is assumed if **week_day_range_list** is omitted,
- syntax and semantics of **daytime_range_list** and **state** are identical to the definition given for the year field above,
- the queue is assumed to be enabled for days neither referenced implicitly (by omitting the **week_day_range_list**) nor explicitly

and where **week_day_range_list** is defined as

```

week_day_range_list := {weekday-weekday | weekday},...
week_day := {mon | tue | wed | thu | fri | sat | sun}

```

SEMANTICS

Successive entries to the **year** and **week** fields (separated by blanks) are combined in compliance with the following rule:

- “off”-areas are overridden by overlapping “on”- and “suspended”-areas.

Hence an entry of the form

week 12-18 tue=13-17=on

means that queues referencing the corresponding calendar are disabled the entire week with the exception of Tuesday between 13.00-17.00 where the queues are available.

EXAMPLES

(The following examples are contained in the directory \$CODINE_ROOT/util/resources/calendars).

- Night, weekend and public holiday calendar:

On public holidays “night” queues are explicitly enabled. On working days queues are disabled between 6.00 and 20.00. Saturday and Sunday are implicitly handled as enabled times:

```
calendar_name  night
year           1.1.1999,6.1.1999,28.3.1999,30.3.1999-
               31.3.1999,18.5.1999-19.5.1999,3.10.1999,25.12.1999,26.12.1999=on
week           mon-fri=6-20
```

- Day calendar:

On public holidays “day”-queues are disabled. On working days such queues are closed during the night between 20.00 and 6.00, i.e. the queues are also closed on Monday from 0.00 to 6.00 and on Friday from 20.00 to 24.00. On Saturday and Sunday the queues are disabled.

```
calendar_name  day
year           1.1.1999,6.1.1999,28.3.1999,30.3.1999-
               31.3.1999,18.5.1999-19.5.1999,3.10.1999,25.12.1999,26.12.1999
week           mon-fri=20-6 sat-sun
```

❑ Night, weekend and public holiday calendar with suspension:

Essentially the same scenario as the first example but queues are suspended instead of switching them “off”.

```
calendar_name  night_s
year           1.1.1999,6.1.1999,28.3.1999,30.3.1999-
31.3.1999,18.5.1999-19.5.1999,3.10.1999,25.12.1999,26.12.1999=on
week           mon-fri=6-20=suspended
```

❑ Day calendar with suspension:

Essentially the same scenario as the second example but queues are suspended instead of switching them “off”.

```
calendar_name  day_s
year           1.1.1999,6.1.1999,28.3.1999,30.3.1999-
31.3.1999,18.5.1999-19.5.1999,3.10.1999,25.12.1999,26.12.1999=suspended
week           mon-fri=206=suspended sat-sun=suspended
```

SEE ALSO

sge_intro(1), *qconf(1)*, *queue_conf(5)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

checkpoint – Sun Grid Engine checkpointing environment configuration file format

DESCRIPTION

Checkpointing is a facility to save the complete status of an executing program or job and to restore and restart from this so called checkpoint at a later point of time if the original program or job was halted, e.g. through a system crash.

Sun Grid Engine provides various levels of checkpointing support (see *sge_ckpt(1)*). The checkpointing environment described here is a means to configure the different types of checkpointing in use for your Sun Grid Engine cluster or parts thereof. For that purpose you can define the operations which have to be executed in initiating a checkpoint generation, a migration of a checkpoint to another host or a restart of a checkpointed application as well as the list of queues which are eligible for a checkpointing method.

Supporting different operating systems may easily force Sun Grid Engine to introduce operating system dependencies for the configuration of the checkpointing configuration file and updates of the supported operating system versions may lead to frequently changing implementation details. Please refer to the file `<codine_root>/doc/checkpointing.asc` for more information.

Please use the `-ackpt`, `-dckpt`, `-mckpt` or `-sckpt` options to the *qconf(1)* command to manipulate checkpointing environments from the command-line or use the corresponding *qmon(1)* dialogue for X-Windows based interactive configuration.

FORMAT

The format of a *checkpoint* file is defined as follows:

ckpt_name

The name of the checkpointing environment. To be used in the *qsub(1)* **-ckpt** switch or for the *qconf(1)* options mentioned above.

interface

The type of checkpointing to be used. Currently, the following types are valid:

hibernator

The Hibernator kernel level checkpointing is interfaced.

cpr

The SGI kernel level checkpointing is used.

cray-ckpt

The Cray kernel level checkpointing is assumed.

transparent

Sun Grid Engine assumes that the jobs submitted with reference to this checkpointing interface use a checkpointing library such as provided by the public domain package *Condor*.

userdefined

Sun Grid Engine assumes that the jobs submitted with reference to this checkpointing interface perform their private checkpointing method.

application-level

Uses all of the interface commands configured in the checkpointing object like in the case of one of the kernel level checkpointing interfaces (*cpr*, *cray-ckpt*, etc.) except for the **restart_command** (see below), which is not used (even if it is configured) but the job script is invoked in case of a restart instead.

queue_list

A comma separated list of queues to which parallel jobs belonging to this parallel environment have access to.

ckpt_command

A command-line type command string to be executed by Sun Grid Engine in order to initiate a checkpoint.

migr_command

A command-line type command string to be executed by Sun Grid Engine during a migration of a checkpointing job from one host to another.

restart_command

A command-line type command string to be executed by Sun Grid Engine when restarting a previously checkpointed application.

clean_command

A command-line type command string to be executed by Sun Grid Engine in order to cleanup after a checkpointed application has finished.

ckpt_dir

A file system location to which checkpoints of potentially considerable size should be stored.

queue_list

Contains a comma or blank separated list of queue names which are eligible for a job if the checkpointing environment was specified at the submission of the job.

ckpt_signal

A Unix signal to be sent to a job by Sun Grid Engine to initiate a checkpoint generation. The value for this field can either be a symbolic name from the list produced by the *-l* option of the *kill(1)* command or an integer number which must be a valid signal on the systems used for checkpointing.

when

The points of time when checkpoints are expected to be generated. Valid values for this parameter are composed by the letters *s*, *m* and *x* and any combinations thereof without any separating character in between. The same letters are allowed for the *-c* option of the *qsub(1)* command which will overwrite the definitions in the used checkpointing environment. The meaning of the letters is defined as follows:

s

A job is checkpointed, aborted and if possible migrated if the corresponding *cod_execd(8)* is shut down on the job's machine.

m

Checkpoints are generated periodically at the *min_cpu_interval* interval defined by the queue (see *queue_conf(5)*) in which a job executes.

x

A job is checkpointed, aborted and if possible migrated as soon as the job gets suspended (manually as well as automatically).

RESTRICTIONS

Note – The functionality of any checkpointing, migration or restart procedures provided by default with the Sun Grid Engine distribution as well as the way how they are invoked in the *ckpt_command*, *migr_command* or *restart_command* parameters of any default checkpointing environments should not be changed or otherwise the functionality remains the full responsibility of the administrator configuring the checkpointing environment. Sun Grid Engine will just invoke these procedures and evaluate their exit status. If the procedures do not perform their tasks properly or are not invoked in a proper fashion, the checkpointing mechanism may behave unexpectedly, Sun Grid Engine has no means to detect this.

SEE ALSO

sge_intro(1), *sge_ckpt(1)*, *qconf(1)*, *qmod(1)*, *qsub(1)*, *cod_execd(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

cod_request – Sun Grid Engine default request definition file format

DESCRIPTION

cod_request reflects the format of the files to define default request profiles. If available, default request files are read and processed during job submission before any submit options embedded in the job script and before any options in the *qsub(1)* or *qsh(1)* command-line are considered. Thus, the command-line and embedded script options may overwrite the settings in the default request files (see *qsub(1)* or *qsh(1)* for details).

There is a cluster global, a user private and a working directory local default request definition file. The working directory local default request file has the highest precedence and is followed by the user private and then the cluster global default request file.

Note – The *-clear* option to *qsub(1)* or *qsh(1)* can be used to discard any previous settings at any time in a default request file, in the embedded script flags or in a *qsub(1)* or *qsh(1)* command-line option.

The format of the default request definition files is:

- ❑ The default request files may contain an arbitrary number of lines. Blank lines and lines with a '#' sign in the first column are skipped.
- ❑ Each line not to be skipped may contain any *qsub(1)* option as described in the *Sun Grid Engine Reference Manual* Reference Manual. More than one option per line is allowed. The batch script file and argument options to the batch script are not considered as *qsub(1)* options and thus are not allowed in a default request file.

EXAMPLES

The following is a simple example of a default request definition file:

```
# Default Requests File
# request arch to be sun4 and a CPU-time of 5hr
-l arch=sun4,s_cpu=5:0:0
# don't restart the job in case of system crashes
-r n
```


Having defined a default request definition file like this and submitting a job as follows:

```
qsub test.sh
```

would have precisely the same effect as if the job was submitted with:

```
sub -l arch=sun4,s_cpu=5:0:0 -r n test.sh
```

FILES

`<codine_root>/<cell>/common/cod_request`

global defaults file

`$HOME/.cod_request`

user private defaults file

`$cwd/.cod_request`

cwd directory defaults file

SEE ALSO

sge_intro(1), *qsh(1)*, *qsub(1)*, *Sun Grid Engine Installation and Administration Guide*

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

codine_aliases – Sun Grid Engine path aliases file format

DESCRIPTION

The Sun Grid Engine path aliasing facility provides administrators and users with the means to reflect complicated and in-homogeneous file system structures in distributed environments (such as user home directories mounted under different paths on different hosts) and to ensure that Sun Grid Engine is able to locate the appropriate working directories for executing batch jobs.

There is a system global path aliasing file and a user local file . *codine_aliases* defines the format of both:

- ❑ Blank lines and lines with a '#' sign in the first column are skipped.
 - ❑ Each line other than a blank line or a line lead by '#' has to contain four strings separated by any number of blanks or tabs.
 - ❑ The first string specifies a source-path, the second a submit-host, the third an execution-host and the fourth the source-path replacement.
 - ❑ Both the submit- and the execution-host entries may consist of only a '*' sign which matches any host.
- If the *-cwd* flag (and only if – otherwise the user's home directory on the execution host is selected to execute the job) to *qsub(1)* was specified, the path aliasing mechanism is activated and the files are processed as follows:

- ❑ After *qsub(1)* has retrieved the physical current working directory path, the cluster global path aliasing file is read if present. The user path aliases file is read afterwards as if it were appended to the global file.
- ❑ Lines not to be skipped are read from the top of the file one by one while the translations specified by those lines are stored if necessary.
- ❑ A translation is stored only if the submit-host entry matches the host *qsub(1)* is executed on and if the source-path forms the initial part either of the current working directory or of the source-path replacements already stored.
- ❑ As soon as both files are read the stored path aliasing information is passed along with the submitted job.
- ❑ On the execution host, the aliasing information will be evaluated. The leading part of the current working directory will be replaced by the source-path replacement if the execution-host entry of the path alias matches the executing host.

Note – The current working directory string will be changed in this case and subsequent path aliases must match the replaced working directory path to be applied.

EXAMPLES

The following is a simple example of a path aliasing file resolving problems with in-homogeneous paths if *automount(8)* is used:

```
# Path Aliasing File
# src-path  sub-host  exec-host  replacement
/tmp_mnt/  *          *          /
# replaces any occurrence of /tmp_mnt/ by /
# if submitting or executing on any host.
# Thus paths on nfs server and clients are the same
```

FILES

<codine_root>/<cell>/common/codine_aliases
 global aliases file
\$HOME/.codine_aliases user local aliases file

SEE ALSO

sge_intro(1), *qsub(1)*, *Sun Grid Engine Installation and Administration Guide*

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`sgc_conf` – Sun Grid Engine configuration files

DESCRIPTION

sgc_conf defines the global and local Sun Grid Engine configurations and can be shown/modified by *qconf(1)* using the `-sconf/-mconf` options. Only root or the cluster administrator may modify *sgc_conf*.

At its initial start-up, *cod_qmaster(8)* checks to see if a valid Sun Grid Engine configuration is available at a well known location in the Sun Grid Engine internal directory hierarchy. If so, it loads that configuration information and proceeds. If not, *cod_qmaster(8)* writes a generic configuration containing default values to that same location. The Sun Grid Engine execution daemons *cod_execd(8)* upon start-up retrieve their configuration from *cod_qmaster(8)*.

The actual configuration for both *cod_qmaster(8)* and *cod_execd(8)* is a superposition of a so called *global* configuration and a *local* configuration being pertinent for the host on which a master or execution daemon resides. If a local configuration is available, its entries overwrite the corresponding entries of the global configuration.

Note – The local configuration does not have to contain all valid configuration entries, but only those which need to be modified against the global entries.

FORMAT

The paragraphs that follow provide brief descriptions of the individual parameters that compose the global and local configurations for a Sun Grid Engine cluster:

qmaster_spool_dir

The location where the master spool directory resides. Only the *cod_qmaster(8)* needs to have access to this directory. It needs read/write permission for root, however. The master spool directory – in particular the jobs directory and the message log file – may become quite large depending on the size of the cluster and the number of jobs. Be sure to allocate enough disk space and regularly clean off the log files, e.g. via a *cron(8)* job.

Changing the master spool directory will have an effect after restarting *cod_qmaster(8)* only.

The default location for the master spool directory is
<codine_root>/<cell>/spool/qmaster.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

execd_spool_dir

The execution daemon spool directory path. Again, a feasible spool directory requires read/write access permission for root. The entry in the global configuration for this parameter can be overwritten by execution host local configurations, i.e. each *cod_execd(8)* may have a private spool directory with a different path, in which case it needs to provide read/write permission for the root account of the corresponding execution host only.

Under **execd_spool_dir** a directory named corresponding to the unqualified hostname of the execution host is opened and contains all information spooled to disk. Thus, it is possible for the **execd_spool_dirs** of all execution hosts to physically reference the same directory path (the root access restrictions mentioned above need to be met, however).

Changing the execution daemon spool directory will have an effect after restarting *cod_execd(8)* only.

The default location for the execution daemon spool directory is
<codine_root>/<cell>/spool.

The global configuration entry for this value may be overwritten by the execution host local configuration.

qsi_common_dir

The QSI configuration directory path. This directory requires read access permission for root on all hosts running *cod_qstd(8)*. Each *cod_qstd(8)* may have a private spool directory with a different path, in which case it needs to provide read permission for the root account of the corresponding execution host only.

Changing the QSI common directory will have immediate effect for *cod_qstd(8)*.

The default location for the QSI common directory is
<codine_root>/<cell>/common/qsi.

The global configuration entry for this value may be overwritten by the execution host local configuration.

binary_path

The directory path where the Sun Grid Engine binaries reside. It is used within Sun Grid Engine components to locate and startup other Sun Grid Engine programs.

The path name given here is searched for binaries as well as any directory below with a directory name equal to the current operating system architecture. Therefore, /usr/CODINE/bin will work for all architectures, if the corresponding binaries are located in subdirectories named aix43, cray, glinux, hp10, irix6, osf4, solaris, etc.

Each *cod_execd(8)* may have its private binary path. Changing the binary path will have immediate effect for *cod_execd(8)*.

The default location for the binary path is <codine_root>/bin

The global configuration entry for this value may be overwritten by the execution host local configuration.

mailer

mailer is the absolute pathname to the electronic mail delivery agent on your system. It must accept the following syntax:

```
mailer -s <subject-of-mail-message> <recipient>
```

Each *cod_execd(8)* may use a private mail agent. Changing **mailer** will take immediate effect.

The default for **mailer** depends on the operating system of the host on which the Sun Grid Engine master installation was run. Common values are /bin/mail or /usr/bin/Mail.

The global configuration entry for this value may be overwritten by the execution host local configuration.

xterm

xterm is the absolute pathname to the X Window System terminal emulator, *xterm(1)*.

Each *cod_execd(8)* may use a private mail agent. Changing **xterm** will take immediate effect.

The default for **xterm** is /usr/bin/X11/xterm.

The global configuration entry for this value may be overwritten by the execution host local configuration.

load_sensor

A comma separated list of executable shell script paths or programs to be started by *cod_execd(8)* and to be used in order to retrieve site configurable load information (e.g. free space on a certain disk partition).

Each *cod_execd(8)* may use a set of private **load_sensor** programs or scripts. Changing **load_sensor** will take effect after two load report intervalls (see **load_report_time**).

The global configuration entry for this value may be over-written by the execution host local configuration.

In addition to the load sensors configured via **load_sensor**, *cod_execd(8)* searches for an executable file named *qloadsensor* in the execution host's Sun Grid Engine binary directory path. If such a file is found, it is treated like the configurable load sensors defined in **load_sensor**. This facility is intended for pre-installing a default load sensor.

prolog

The executable path of a shell script that is started before execution of Sun Grid Engine jobs with the same environment setting as that for the Sun Grid Engine jobs to be started afterwards. An optional prefix "user@" specifies the user under which this procedure is to be started. This procedure is intended as a means for the Sun Grid Engine administrator to automate the execution of general site specific tasks like the preparation of temporary file systems with the need for the same context information as the job. Each *cod_execd(8)* may use a private prologue script. Correspondingly, the execution host local configurations is can be overwritten by the queue configuration (see *queue_conf(5)*). Changing **prolog** will take immediate effect.

Note – prolog is executed *exactly* as the job script. Therefore, all implications described under the parameters *shell_start_mode* and *login_shells* below apply.

The default for **prolog** is the special value NONE, which prevents from execution of a prologue script.

The following special variables being expanded at runtime can be used (besides any other strings which have to be interpreted by the procedure) to constitute a command line:

\$host

The name of the host on which the prolog or epilog procedures are started.

\$job_owner

The user name of the job owner.

\$job_id

Sun Grid Engine's unique job identification number.

\$job_name

The name of the job.

\$processors

The **processors** string as contained in the queue configuration (see *queue_conf(5)*) of the master queue (the queue in which the prolog and epilog procedures are started).

\$queue

The master queue, i.e. the queue in which the prolog and epilog procedures are started.

The global configuration entry for this value may be overwritten by the execution host local configuration.

epilog

The executable path of a shell script that is started after execution of Sun Grid Engine jobs with the same environment setting as that for the Sun Grid Engine jobs that has just completed. An optional prefix "user@" specifies the user under which this procedure is to be started. This procedure is intended as a means for the Sun Grid Engine administrator to automate the execution of general site specific tasks like the cleaning up of temporary file systems with the need for the same context information as the job. Each *cod_execd(8)* may use a private epilogue script. Correspondingly, the execution host local configurations is can be overwritten by the queue configuration (see *queue_conf(5)*). Changing **epilog** will take immediate effect.

Note – epilog is executed *exactly* as the job script. Therefore, all implications described under the parameters *shell_start_mode* and *login_shells* below apply.

The default for **epilog** is the special value NONE, which prevents from execution of a epilogue script. The same special variables as for **prolog** can be used to constitute a command line.

The global configuration entry for this value may be overwritten by the execution host local configuration.

shell_start_mode

This parameter defines the mechanisms which are used to actually invoke the job scripts on the execution hosts. The following values are recognized:

unix_behavior

If a user starts a job shell script under UNIX interactively by invoking it just with the script name the operating system's executable loader uses the information provided in a comment such as `'#!/bin/csh'` in the first line of the script to detect which command interpreter to start to interpret the script. This mechanism is used by Sun Grid Engine when starting jobs if *unix_behavior* is defined as **shell_start_mode**.

posix_compliant

POSIX does not consider first script line comments such as `'#!/bin/csh'` as being significant. The POSIX standard for batch queuing systems (P1003.2d) therefore requires a compliant queuing system to ignore such lines but to use user specified or configured default command interpreters instead. Thus, if **shell_start_mode** is set to *posix_compliant* Sun Grid Engine will either use the command interpreter indicated by the **-S** option of the *qsub(1)* command or the **shell** parameter of the queue to be used (see *queue_conf(5)* for details).

script_from_stdin

Setting the **shell_start_mode** parameter either to *posix_compliant* or *unix_behavior* requires you to set the umask in use for *cod_execd(8)* such that every user has read access to the *active_jobs* directory in the spool directory of the corresponding execution daemon. In case you have **prolog** and **epilog** scripts configured, they also need to be readable by any user who may execute jobs.

If this violates your site's security policies you may want to set **shell_start_mode** to *script_from_stdin*. This will force Sun Grid Engine to open the job script as well as the epilogue and prologue scripts for reading into STDIN as root (if *cod_execd(8)* was started as root) before changing to the job owner's user account. The script is then fed into the STDIN stream of the command interpreter indicated by the **-S** option of the *qsub(1)* command or the **shell** parameter of the queue to be used (see *queue_conf(5)* for details).

Thus setting **shell_start_mode** to *script_from_stdin* also implies *posix_compliant* behavior.

Note – Feeding scripts into the STDIN stream of a command interpreter may cause trouble if commands like *rsh(1)* are invoked inside a job script as they also process the STDIN stream of the command interpreter. These problems can usually be resolved by redirecting the STDIN channel of those commands to come from `/dev/null` (e.g. `rsh host date < /dev/null`).

Note – Any command-line options associated with the job are passed to the executing shell. The shell will only forward them to the job if they are not recognized as valid shell options.

Changes to **shell_start_mode** will take immediate effect. The default for **shell_start_mode** is *posix_compliant*.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

login_shells

UNIX command interpreters like the Bourne-Shell (see *sh(1)*) or the C-Shell (see *csh(1)*) can be used by Sun Grid Engine to start job scripts. The command interpreters can either be started as login-shells (i.e. all system and user default resource files like *.login* or *.profile* will be executed when the command interpreter is started and the environment for the job will be set up as if the user has just logged in) or just for command execution (i.e. only shell specific resource files like *.cshrc* will be executed and a minimal default environment is set up by Sun Grid Engine – see *qsub(1)*). The parameter **login_shells** contains a comma separated list of the executable names of the command interpreters to be started as login-shells.

Changes to **login_shells** will take immediate effect. The default for **login_shells** is *sh,csh,tcsh,ksh*.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

min_uid

min_uid places a lower bound on user IDs that may use the cluster. Users whose user ID (as returned by *getpwnam(3)*) is less than **min_uid** will not be allowed to run jobs on the cluster.

Changes to **min_uid** will take immediate effect. The default for **min_uid** is 0.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

min_gid

This parameter sets the lower bound on group IDs that may use the cluster. Users whose default group ID (as returned by *getpwnam(3)*) is less than **min_gid** will not be allowed to run jobs on the cluster.

Changes to **min_gid** will take immediate effect. The default for **min_gid** is 0.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

user_lists

The **user_lists** parameter contains a comma separated list of so called user access lists as described in *access_list(5)*. Each user contained in at least one of the enlisted access lists has access to the cluster. If the **user_lists** parameter is set to NONE (the default) any user has access being not explicitly excluded via the **xuser_lists** parameter described below. If a user is contained both in an access list enlisted in **xuser_lists** and **user_lists** the user is denied access to the cluster.

Changes to **user_lists** will take immediate effect

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

xuser_lists

The **xuser_lists** parameter contains a comma separated list of so called user access lists as described in *access_list(5)*. Each user contained in at least one of the enlisted access lists is denied access to the cluster. If the **xuser_lists** parameter is set to NONE (the default) any user has access. If a user is contained both in an access list enlisted in **xuser_lists** and **user_lists** (see above) the user is denied access to the cluster.

Changes to **xuser_lists** will take immediate effect

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

administrator_mail

administrator_mail specifies a comma separated list of the electronic mail address(es) of the cluster administrator(s) to whom internally-generated problem reports are sent. The mail address format depends on your electronic mail system and how it is configured; consult your system's configuration guide for more information.

Changing **administrator_mail** takes immediate effect. The default for **administrator_mail** is an empty mail list.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

projects

This parameter is only available for Sun Grid Engine, Enterprise Edition systems. It is not present in Sun Grid Engine.

The **projects** list contains all projects which are granted access to Sun Grid Engine, Enterprise Edition. User belonging to none of these projects cannot use Sun Grid Engine, Enterprise Edition. If users belong to projects in the **projects** list and the **xprojects** list (see below), they also cannot use the system.

Changing **projects** takes immediate effect. The default for **projects** is none.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

xprojects

This parameter is only available for Sun Grid Engine, Enterprise Edition systems. It is not present in Sun Grid Engine.

The **xprojects** list contains all projects which are granted access to Sun Grid Engine, Enterprise Edition. User belonging to one of these projects cannot use Sun Grid Engine, Enterprise Edition. If users belong to projects in the **projects** list (see above) and the **xprojects** list, they also cannot use the system.

Changing **xprojects** takes immediate effect. The default for **xprojects** is none.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

load_report_time

System load is reported periodically by the execution daemons to *cod_qmaster(8)*. The parameter **load_report_time** defines the time interval between load reports.

Each *cod_execd(8)* may use a different load report time. Changing **load_report_time** will take immediate effect.

Note – Be careful when modifying **load_report_time**. Reporting load too frequently might block *cod_qmaster(8)* especially if the number of execution hosts is large. Moreover, since the system load typically increases and decreases smoothly, frequent load reports hardly offer any benefit.

The default for **load_report_time** is 40 seconds.

The global configuration entry for this value may be overwritten by the execution host local configuration.

stat_log_time

Sun Grid Engine periodically logs a snapshot of the status of the queues currently configured in the cluster to disk. The time interval in seconds between consecutive snapshots is defined by **stat_log_time**.

Changing **stat_log_time** takes immediate effect. The default for **stat_log_time** is 2 hours 30 minutes.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

max_unheard

If *cod_qmaster(8)* could not contact or was not contacted by the execution daemon of a host for **max_unheard** seconds, all queues residing on that particular host are set to status unknown. *cod_qmaster(8)*, at least, should be contacted by the execution daemons in order to get the load reports. Thus, **max_unheard** should be greater than the **load_report_time** (see above).

Changing **max_unheard** takes immediate effect. The default for **max_unheard** is 2 minutes 30 seconds.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

loglevel

This parameter specifies the level of detail that Sun Grid Engine components such as *cod_qmaster(8)* or *cod_execd(8)* use to produce informative, warning or error messages which are logged to the *messages* files in the master and execution daemon spool directories (see the description of the **qmaster_spool_dir** and the **execd_spool_dir** parameter above). The following message levels are available:

log_err

All error events being recognized are logged.

log_warning

All error events being recognized and all detected signs of potentially erroneous behavior are logged.

log_info

All error events being recognized, all detected signs of potentially erroneous behavior and a variety of informative messages are logged.

Changing **loglevel** will take immediate effect.

The default for **loglevel** is *log_info*.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

enforce_project

This parameter is only available for Sun Grid Engine, Enterprise Edition systems. It is not present in Sun Grid Engine.

If set to *true*, **users are required to request a project whenever submitting a job.** See the **-P** option to *qsub(1)* for details.

Changing **enforce_project** will take immediate effect. The default for **enforce_project** is *false*.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

set_token_cmd

This parameter is only present if your Sun Grid Engine system is licensed to support AFS.

Set_token_cmd points to a command which sets and extends AFS tokens for Sun Grid Engine jobs. In the standard Sun Grid Engine AFS distribution, it is supplied as a script which expects two command line parameters. It reads the token from STDIN, extends the token's expiration time and sets the token:

```
<set_token_cmd> <user> <token_extend_after_seconds>
```

As a shell script this command will call the programs:

- SetToken
- forge

which are provided by your distributor as source code. The script looks as follows:

```
#!/bin/sh
# set_token_cmd
forge -u $1 -t $2 | SetToken
```

Since it is necessary for *forge* to read the secret AFS server key, a site might wish to replace the **set_token_cmd** script by a command, which connects to a self written daemon at the AFS server. The token must be forged at the AFS server and returned to the local machine, where *SetToken* is executed.

Changing **set_token_cmd** will take immediate effect. The default for **set_token_cmd** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

pag_cmd

This parameter is only present if your Sun Grid Engine system is licensed to support AFS.

The path to your *pagsh* **is specified via this parameter. The *cod_shepherd(8)*** process and the job run in a *pagsh*. Please ask your AFS administrator for details.

Changing **pag_cmd** will take immediate effect. The default for **pag_cmd** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

token_extend_time

This parameter is only present if your Sun Grid Engine system is licensed to support AFS.

the time period for which AFS tokens are periodically extended. Sun Grid Engine will call the token extension 30 minutes before the tokens expire until jobs have finished and the corresponding tokens are no longer required.

Changing **token_extend_time** will take immediate effect. The default for **token_extend_time** is 24:0:0, i.e. 24 hours.

The global configuration entry for this value may be overwritten by the execution host local configuration.

gid_range

This parameter is only available for Sun Grid Engine, Enterprise Edition systems. It is not present in Sun Grid Engine.

The **gid_range** is a comma separated list of range expressions of the form n-m (n as well as m being positive non-zero integer numbers), where m is an abbreviation for m-m. These numbers are used in *cod_execd(8)* to identify processes belonging to the same job.

Each *cod_execd(8)* may use a separate set up group ids for this purpose. All number in the group id range have to be unused supplementary group ids on the system, where the *cod_execd(8)* is started.

Changing **gid_range** will take immediate effect. There is no default for **gid_range**. The administrator will have to assign a value for **gid_range** during installation of Sun Grid Engine, Enterprise Edition.

The global configuration entry for this value may be overwritten by the execution host local configuration.

qmaster_params

A list of additional parameters can be passed to the Sun Grid Engine qmaster. The following values are recognized:

ENABLE_FORCED_QDEL

If this parameter is set, non-administrative users can force deletion of their own jobs via the *-f* option of *qdel(1)*. Without this parameter, forced deletion of jobs is only allowed by the Sun Grid Engine manager or operator.

Note – Forced deletion for jobs is executed differently depending on whether users are Sun Grid Engine administrators or not. In case of administrative users, the jobs are removed from the internal database of Sun Grid Engine immediately. For regular users, the equivalent of a normal *qdel(1)* is executed first, and deletion is forced only if the normal cancellation was unsuccessful.

Changing **qmaster_params** will take immediate effect. The default for **qmaster_params** is none.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

FORBID_RESCHEDULE

If this parameter is set, re-queuing of jobs cannot be initiated by the job script which is under control of the user. Without this parameter jobs returning the value 99 are rescheduled. This can be used to cause the job to be restarted at a different machine, for instance if there are not enough resources on the current one.

Changing **qmaster_params** will take immediate effect. The default for **qmaster_params** is none.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

schedd_params

This is foreseen for passing additional parameters to the Sun Grid Engine scheduler. The following values are recognized currently:

FLUSH_SUBMIT_SEC, FLUSH_FINISH_SEC

The parameters are provided for tuning the system's scheduling behavior. By default, a scheduler run is triggered in the scheduler interval which is defined in the scheduler configuration *sched_conf(5)*, parameter **schedule_interval**.

The parameters FLUSH_SUBMIT_SEC/FLUSH_FINISH_SEC define the time gaps between triggering a scheduler run and the submitting/finishing of a job.

The most immediate scheduler reaction can be activated by setting both values to 0. The default scheduling behavior is enforced by either removing the parameters or setting them to a value of -1.

Changing **schedd_params** will take immediate effect. The default for **schedd_params** is none.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

execd_params

This is foreseen for passing additional parameters to the Sun Grid Engine execution daemon. The following values are recognized:

ACCT_RESERVED_USAGE

If this parameter is set to true, for reserved usage is used for the accounting entries **cpu**, **mem** and **io** instead of the measured usage.

KEEP_ACTIVE

This value should only be set for debugging purposes. If set to true, the execution daemon will not remove the spool directory maintained by *cod_shepherd(8)* for a job.

PTF_MIN_PRIORITY, PTF_MAX_PRIORITY

The parameters are only available in a Sun Grid Engine, Enterprise Edition system.

The maximum/minimum priority which Sun Grid Engine, Enterprise Edition will assign to a job. Typically this is a negative/postive value in the range of -20 (maximum) to 19 (minimum) for systems which allow setting of priorities with the *nice(2)* system call. Other systems may provide different ranges.

The default priority range (varies from system to system) is installed either by removing the parameters or by setting a value of -999.

See the "messages" file of the execution daemon for the predefined default value on your hosts. The values are logged during the startup of the execution daemon.

NOTIFY_KILL

The parameter allows you to change the notification signal for the signal SIGKILL (see *-notify* option of *qsub(1)*). The parameter either accepts signal names (use the *-l* option of *kill(1)*) or the special value *none*. If set to *none*, no notification signal will be sent. If it is set to *TERM*, for instance, or another signal name then this signal will be sent as notification signal.

NOTIFY_SUSP

With this parameter it is possible to modify the notification signal for the signal SIGSTOP (see *-notify* parameter of *qsub(1)*). The parameter either accepts signal names (use the *-l* option of *kill(1)*) or the special value *none*. If set to *none*, no notification signal will be sent. If it is set to *TSTP*, for instance, or another signal name then this signal will be sent as notification signal.

SHARETREE_RESERVED_USAGE

If this parameter is set to true, reserved usage is taken for the Sun Grid Engine, Enterprise Edition share tree consumption instead of measured usage.

USE_QSUB_GID

If this parameter is set to true, the primary group id being active when a job was submitted will be set to become the primary group id for job execution. If the parameter is not set, the primary group id as defined for the job owner in the execution host *passwd(5)* file is used.

Changing **execd_params** will take immediate effect. The default for **execd_params** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

admin_user

Administrative user account used by Sun Grid Engine for all internal file handling (status spooling, message logging, etc.). Can be used in cases where the root account does not have the corresponding file access permissions (e.g. on a shared file system without global root read/write access).

Changing **admin_user** will take immediate effect, but if access to the Sun Grid Engine spooling area is interrupted, this will result in unpredictable behavior. The **admin_user** parameter has no default value, but instead it is defined during the master installation procedure.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

finished_jobs

Sun Grid Engine stores a certain number of *just finished* jobs to provide post mortem status information. The **finished_jobs** parameter defines the number of finished jobs being stored. If this maximum number is reached, the eldest finished job will be discarded for every new job being added to the finished job list.

Changing **finished_jobs** will take immediate effect. The default for **finished_jobs** is 0.

This value is a global configuration parameter only. It cannot be overwritten by the execution host local configuration.

qlogin_daemon

This parameter specifies the executable that is to be started on the server side of a *qlogin(1)* request. Usually this is the fully qualified pathname of the system's telnet daemon. If no value is given, a specialized Sun Grid Engine component is used.

Changing **qlogin_daemon** will take immediate effect. The default for **qlogin_daemon** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

qlogin_command

This is the command to be executed on the client side of a *qlogin(1)* request. Usually this is the fully qualified pathname of the system's telnet client program. If no value is given, a specialized Sun Grid Engine component is used. It is automatically started with the target host and port number as parameters.

Changing **qlogin_command** will take immediate effect. The default for **qlogin_command** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

rlogin_daemon

This parameter specifies the executable that is to be started on the server side of a *qrsh(1)* request **without** a command argument to be executed remotely. Usually this is the fully qualified pathname of the system's rlogin daemon. If no value is given, a specialized Sun Grid Engine component is used.

Changing **rlogin_daemon** will take immediate effect. The default for **rlogin_daemon** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

rlogin_command

This is the command to be executed on the client side of a *qrsh(1)* request **without** a command argument to be executed remotely. Usually this is the fully qualified pathname of the system's rlogin client program. If no value is given, a specialized Sun Grid Engine component is used. The command is automatically started with the target host and port number as parameters like required for

telnet(1). The Sun Grid Engine rlogin client has been extended to accept and use the port number argument. You can only use clients, such as *ssh*, which also understand this syntax.

Changing **rlogin_command** will take immediate effect. The default for **rlogin_command** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

rsh_daemon

This parameter specifies the executable that is to be started on the server side of a *qrsh(1)* request **with** a command argument to be executed remotely. Usually this is the fully qualified pathname of the system's rsh daemon. If no value is given, a specialized Sun Grid Engine component is used.

Changing **rsh_daemon** will take immediate effect. The default for **rsh_daemon** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

rsh_command

This is the command to be executed on the client side of a *qrsh(1)* request **with** a command argument to be executed remotely. Usually this is the fully qualified pathname of the system's rsh client program. If no value is given, a specialized Sun Grid Engine component is used. The command is automatically started with the target host and port number as parameters like required for *telnet(1)* plus the command with its arguments to be executed remotely. The Sun Grid Engine rsh client has been extended to accept and use the port number argument. You can only use clients, such as *ssh*, which also understand this syntax.

Changing **rsh_command** will take immediate effect. The default for **rsh_command** is none.

The global configuration entry for this value may be overwritten by the execution host local configuration.

default_domain

Should be set if your hostname resolving yields unqualified hostnames for your cluster hosts. In that case, the value of **default_domain** is appended to the unqualified hostname to define a fully qualified hostname.

Changing **default_domain** will take immediate effect. The default for **default_domain** is "none", in which case it will not be used.

The global configuration entry for this value may be overwritten by the execution host local configuration.

SEE ALSO

sgc_intro(1), csh(1), qconf(1), qsub(1), rsh(1), sh(1), getpwnam(3), queue_conf(5), sched_conf(5), cod_execd(8), cod_qmaster(8), cod_shepherd(8), cron(8), Sun Grid Engine Installation and Administration Guide.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

SGE_H_ALIASES(5)

NAME

`sgc_h_aliases` – Sun Grid Engine host aliases file format

DESCRIPTION

All Sun Grid Engine components use a hostname resolving service provided by *cod_commd(5)* to identify hosts via a unique hostname. *cod_commd(5)* itself references standard UNIX directory services such as **DNS**, **NIS** and `/etc/hosts` to resolve hostnames. In rare cases these standard services cannot be setup cleanly and Sun Grid Engine communication daemons running on different hosts are unable to automatically determine a unique hostname for one or all hosts which can be used on all hosts. In such situations a Sun Grid Engine host aliases file can be used to provide the communication daemons with a private and consistent hostname resolution database.

If a host aliases file is used, it must be specified explicitly to *cod_commd(8)* via the **-a** command line option.

FORMAT

For each host a single line must be provided with a blank, comma or semicolon separated list of hostname aliases. The first alias is defined to be the **unique** hostname which will be used by all Sun Grid Engine components using the hostname aliasing service of the *cod_commd(8)*.

SEE ALSO

sgc_intro(1), *cod_commd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

`sge_pe` – Sun Grid Engine parallel environment configuration file format

DESCRIPTION

Parallel environments are parallel programming and runtime environments allowing for the execution of shared memory or distributed memory parallelized applications. Parallel environments usually require some kind of setup to be operational before starting parallel applications. Examples for common parallel environments are shared memory parallel operating systems and the distributed memory environments Parallel Virtual Machine (PVM) or Message Passing Interface (MPI).

`sge_pe` allows for the definition of interfaces to arbitrary parallel environments. Once a parallel environment is defined or modified with the **-ap** or **-mp** options to `qconf(1)` the environment can be requested for a job via the **-pe** switch to `qsub(1)` together with a request of a range for the number of parallel process to be allocated by the job. Additional **-l** options may be used to specify the job requirement to further detail.

FORMAT

The format of a `sge_pe` file is defined as follows:

pe_name

The name of the parallel environment. To be used in the `qsub(1)` **-pe** switch.

queue_list

A comma separated list of queues to which parallel jobs belonging to this parallel environment have access to.

slots

The number of parallel processes being allowed to run in total under the parallel environment concurrently.

user_lists

A comma separated list of user access list names (see *access_list(5)*). Each user contained in at least one of the enlisted access lists has access to the parallel environment. If the **user_lists** parameter is set to NONE (the default) any user has access being not explicitly excluded via the **xuser_lists** parameter described below. If a user is contained both in an access list enlisted in **xuser_lists** and **user_lists** the user is denied access to the parallel environment.

xuser_lists

The **xuser_lists** parameter contains a comma separated list of so called user access lists as described in *access_list(5)*. Each user contained in at least one of the enlisted access lists is not allowed to access the parallel environment. If the **xuser_lists** parameter is set to NONE (the default) any user has access. If a user is contained both in an access list enlisted in **xuser_lists** and **user_lists** the user is denied access to the parallel environment.

start_proc_args

The invocation command line of a start-up procedure for the parallel environment. The start-up procedure is invoked by *cod_shepherd(8)* prior to executing the job script. Its purpose is to setup the parallel environment correspondingly to its needs. An optional prefix “user@” specifies the user under which this procedure is to be started. The standard output of the start-up procedure is redirected to the file *REQNAME.poJID* in the job’s working directory (see *qsub(1)*), with *REQNAME* being the name of the job as displayed by *qstat(1)* and *JID* being the job’s identification number. Likewise, the standard error output is redirected to *REQNAME.peJID*

The following special variables being expanded at runtime can be used (besides any other strings which have to be interpreted by the start and stop procedures) to constitute a command line:

\$pe_hostfile

The pathname of a file containing a detailed description of the layout of the parallel environment to be setup by the start-up procedure. Each line of the file refers to a host on which parallel processes are to be run. The first entry of each line denotes the hostname, the second entry the number of parallel processes to be run on the host and the third entry a processor range to be used in case of a multiprocessor machines.

\$host

The name of the host on which the start-up or stop procedures are started.

\$job_owner

The user name of the job owner.

\$job_id

Sun Grid Engine's unique job identification number.

\$job_name

The name of the job.

\$pe

The name of the parallel environment in use.

\$pe_slots

Number of slots granted for the job.

\$processors

The **processors** string as contained in the queue configuration (see *queue_conf(5)*) of the master queue (the queue in which the start-up and stop procedures are started).

\$queue

The master queue, i.e. the queue in which the start-up and stop procedures are started.

stop_proc_args

The invocation command line of a shutdown procedure for the parallel environment. The shutdown procedure is invoked by *cod_shepherd(8)* after the job script has finished. Its purpose is to stop the parallel environment and to remove it from all participating systems. An optional prefix "user@" specifies the user under which this procedure is to be started. The standard output of the stop procedure is also redirected to the file *REQNAME.poJID* in the job's working directory (see *qsub(1)*), with *REQNAME* being the name of the job as displayed by *qstat(1)* and *JID* being the job's identification number. Likewise, the standard error output is redirected to *REQNAME.peJID*.

The same special variables as for **start_proc_args** can be used to constitute a command line.

signal_proc_args

The invocation command line of a signalling procedure for the parallel environment. The signalling procedure is invoked by *cod_shepherd(8)* after whenever a signal is sent to the parallel job via *qmod(1)*, *qdel(1)* or in case of a migration request. Its purpose is to signal all components of the parallel environment and their associated application processes correspondingly. The standard output of the signalling procedure is also redirected to the file *REQNAME.poJID* in the job's working directory (see *qsub(1)*), with *REQNAME*

being the name of the job as displayed by *qstat(1)* and *JID* being the job's identification number. Likewise, the standard error output is redirected to *REQNAME.pe/JID*

The same special variables as for **start_proc_args** can be used to constitute a command line.

allocation_rule

The allocation rule is interpreted by *cod_schedd(8)* and helps the scheduler to decide how to distribute parallel processes among the available machines. If, for instance, a parallel environment is built for shared memory applications only, all parallel processes have to be assigned to a single machine, no matter how much suitable machines are available. If, however, the parallel environment follows the distributed memory paradigm, an even distribution of processes among machines may be favorable.

The current version of the scheduler only understands the following allocation rules:

<int>:

An integer number fixing the number of processes per host. If the number is 1, all processes have to reside on different hosts. If the special denominator **\$pe_slots** is used, the full range of processes as specified with the *qsub(1)* **-pe** switch has to be allocated on a single host (no matter which value belonging to the range is finally chosen for the job to be allocated).

\$fill_up:

Starting from the best suitable host/queue, all available slots are allocated. Further hosts and queues are "filled up" as long as a job still requires slots for parallel tasks.

\$round_robin:

From all suitable hosts a single slot is allocated until all tasks requested by the parallel job are dispatched. If more tasks are requested than suitable hosts are found, allocation starts again from the first host. The allocation scheme walks through suitable hosts in a best-suitable-first order.

control_slaves

This parameter can be set to TRUE or FALSE (the default). It indicates whether Sun Grid Engine is the creator of the slave tasks of a parallel application via *cod_execd(8)* and *cod_shepherd(8)* and thus has full control over all processes in a parallel application, which enables capabilities such as resource limitation and correct accounting. However, to gain control over the slave tasks of a parallel application, a sophisticated PE interface is required, which works closely together with Sun Grid Engine facilities. Such PE interfaces are available through your local Sun Grid Engine support office.

Please set the *control_slaves* parameter to false for all other PE interfaces.

job_is_first_task

This parameter is only checked if **control_slaves** (see above) is set to TRUE and thus Sun Grid Engine is the creator of the slave tasks of a parallel application via *cod_execd(8)* and *cod_shepherd(8)*. In this case, a sophisticated PE interface is required closely coupling the parallel environment and Sun Grid Engine. The documentation accompanying such PE interfaces will recommend the setting for **job_is_first_task**.

The **job_is_first_task** parameter can be set to TRUE or FALSE. A value of TRUE indicates that the Sun Grid Engine job script already contains one of the tasks of the parallel application, while a value of FALSE indicates that the job script (and its child processes) is not part of the parallel program.

RESTRICTIONS

Note – The functionality of the start-up, shutdown and signalling procedures remains the full responsibility of the administrator configuring the parallel environment. Sun Grid Engine will just invoke these procedures and evaluate their exit status. If the procedures do not perform their tasks properly or if the parallel environment or the parallel application behave unexpectedly, Sun Grid Engine has no means to detect this.

SEE ALSO

sgc_intro(1), *qconf(1)*, *qdel(1)*, *qmod(1)*, *qsub(1)*, *access_list(5)*, *cod_qmaster(8)*, *cod_schedd(8)*, *cod_shepherd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

complex – Sun Grid Engine complexes configuration file format

DESCRIPTION

Complex reflects the format of the Sun Grid Engine complexes configuration. The definition of complexes provides all pertinent information concerning the resource attributes a user may request for a Sun Grid Engine job via the *qsub(1)* **-I** option and for the interpretation of these parameters within the Sun Grid Engine system.

The complexes configuration files should not be accessed directly. In order to add or modify complexes, the *qconf(1)* options **-Ac**, **-ac**, **-Mc** and **-mc** should be used instead. While the **-Ac** and **-Mc** options take a *complex* configuration file as an argument, the **-ac** and **-mc** options bring up an editor filled in with a template *complex* configuration or the configuration of an existing complex.

The Sun Grid Engine complexes object integrates 4 different types of complexes:

The Queue Complex

It is referenced by the special name “queue”.

In its default form it contains a selection of parameters in the queue configuration as defined in *queue_conf(5)*. The queue configuration parameters being requestable for a job by the user in principal are:

- qname
- hostname
- priority
- notify
- calendar
- max_migr_time
- max_no_migr
- min_cpu_interval
- tmpdir
- seq_no
- s_rt
- h_rt

s_cpu
h_cpu
s_data
h_data
s_stack
h_stack
s_core
h_core
s_rss
h_rss

The queue complex can be extended if further attributes are intended to be available for each queue. The queue complex defines the characteristics (such as the data type) of the attributes it contains. A value setting for the queue complex attributes is defined by the queue configuration for each queue in case of the standard parameters enlisted above, or by the **complex_values** entry in the queue configuration (see *queue_conf(5)* for details) if a parameter has been added to the default queue complex. If no definition for the value in the **complex_values** entry of the queue configuration is given in the latter case, the value is set as defined by the **value** field described below.

The Host Complex

It is referenced by the special name “host” and contains the characteristics definition of all attributes which are intended to be managed on a host basis. The standard set of host related attributes consists of two categories, but it may be enhanced like the queue complex as described above. The first category is built by several queue configuration attributes which are particularly suitable to be managed on a host basis. These attributes are:

slots
s_vmem
h_vmem
s_fsize
h_fsize

(please refer to *queue_conf(5)* for details).

Note – Defining these attributes in the host complex is no contradiction to having them also in the queue configuration. It allows maintaining the corresponding resources on a host level and at the same time on a queue level. Total virtual free memory (h_vmem) can be managed for a host, for example, and a subset of the total amount can be associated with a queue on that host.

The second attribute category in the standard host complex are the default load values. Every *cod_execd(8)* periodically reports load to *cod_qmaster(8)*. The reported load values are either the standard Sun Grid Engine load values such as the CPU load average (see *uptime(1)*) or load values defined by the Sun Grid Engine administration (see the **load_sensor** parameter in the cluster configuration *sge_conf(5)* and the *Sun Grid Engine Installation and Administration Guide* for details). The characteristics definition for the standard load values is part of the default host complex, while administrator defined load values require extension of the host complex. Please refer to the file <codine_root>/doc/load_parameters.asc for detailed information on the standard set of load values.

The host complex commonly is not only extended to include non-standard load parameters, but also to manage host related resources such as the number of software licenses being assigned to a host or the available disk space on a host local filesystem.

A concrete value for a particular host complex attribute is determined by either an associated queue configuration in the case of the queue configuration derived attributes, a reported load value or the explicit definition of a value in the **complex_values** entry of the corresponding host configuration (see *host_conf(5)*). If none of the above is available (e.g. the value is supposed to be a load parameter, but *cod_execd(8)* does not report a load value for it), the **value** field described below is used.

The Global Complex

It is referenced by the special name “global”.

The entries configured in the global complex refer to cluster wide resource attributes, such as the number of available “floating” licenses of a particular software or the free disk space on a network wide available filesystem. Global resource attributes can also be associated with load reports, if the corresponding load report contains the “GLOBAL” identifier (see the corresponding section in the *Sun Grid Engine Installation and Administration Guide* for details). Global load values can be reported from any host in the cluster. There are no global load values reported by Sun Grid Engine by default and hence there is no default global complex configuration.

Concrete values for global complex attributes are either determined by global load reports or by explicit definition in the **complex_values** parameter of the “global” host configuration (see *host_conf(5)*). If none of both is present (e.g. a load value has not yet been reported) the **value** field described below is used.

User Defined Complexes

By setting up user defined complexes the Sun Grid Engine administration has the ability to extend the set of attributes managed by Sun Grid Engine while restricting the influence of those attributes to particular queues and/or hosts. A user complex is just a named collection of attributes and the corresponding definition as to how these attributes are to be handled by Sun Grid Engine. One or more of these user defined complexes can be attached to a queue and/ or host via the **complex_list** queue and host configuration parameter (see *queue_conf(5)* and *host_conf(5)*). The attributes defined in all assigned complexes become available to the queue and the host respectively in addition to the default complex attributes.

Concrete values for user defined complexes have to be set by the **complex_values** parameter in the queue and host configuration or otherwise the **value** field described below is used.

FORMAT

The principal format of a *complex* configuration is that of a tabulated list. Each line starting with a '#' character is a comment line. Each line despite comment lines define one element of the complex. A element definition line consists of the following 6 column entries per line (in the order of appearance):

name

The name of the complex element to be used to request this attribute for a job in the *qsub(1)* **-I** option. An attribute **name** may appear only once across all complexes, i.e. the complex attribute definition is unique.

shortcut

A shortcut for **name** which may also be used to request this attribute for a job in the *qsub(1)* **-I** option. An attribute **shortcut** may appear only once across all complexes, so as to avoid the possibility of ambiguous complex attribute references.

type

This setting determines how the corresponding values are to be treated Sun Grid Engine internally in case of comparisons or in case of load scaling for the load complex entries:

- With **INT** only raw integers are allowed.
- With **DOUBLE** floating point numbers in double precision (decimal and scientific notation) can be specified.

- With **TIME** time specifiers are allowed. Refer to *queue_conf(5)* for a format description.
- With **MEMORY** memory size specifiers are allowed. Refer to *queue_conf(5)* for a format description.
- With **BOOL** the strings TRUE and FALSE are allowed. When used in a load formula (refer to *sched_conf(5)*) TRUE and FALSE get mapped into '1' and '0'.
- With **STRING** all strings are allowed and *strcmp(3)* is used for comparisons.
- **CSTRING** is like **STRING** except comparisons are case insensitive.
- **HOST** is like **CSTRING** but the string must be a valid hostname.

value

The **value** field is a pre-defined value setting for an attribute, which only has an effect if it is not overwritten while attempting to determine a concrete value for the attribute with respect to a queue, a host or the Sun Grid Engine cluster. The value field can be overwritten by

- the queue configuration values of a referenced queue.
- host specific and cluster related load values.
- explicit specification of a value via the *complex_values* parameter in the queue or host configuration (see *queue_conf(5)* and *host_conf(5)* for details).

If none of above is applicable, value is set for the attribute.

relop

The **relation operator**. The relation operator is used when the value requested by the user for this parameter is compared against the corresponding value configured for the considered queues. If the result of the comparison is false, the job cannot run in this queue. Possible relation operators are "=", "<", ">", "<=" and ">=". The only valid operator for string type attributes is "=".

requestable

The entry can be used in a *qsub(1)* resource request if this field is set to 'y' or 'yes'. If set to 'n' or 'no' this entry cannot be used by a user in order to request a queue or a class of queues. If the entry is set to 'forced' or 'f' the attribute has to be requested by a job or it is rejected.

consumable

The **consumable** parameter can be set to either 'yes' ('y' abbreviated) or 'no' ('n'). It can be set to 'yes' only for numeric attributes (INT, MEMORY, TIME - see **type** above). If set to 'yes' the consumption of the corresponding resource can be managed by Sun Grid Engine internal bookkeeping. In this case Sun Grid Engine accounts for the consumption of this resource for all running jobs and ensures that jobs are only dispatched if the Sun Grid Engine internal bookkeeping

indicates enough available consumable resources. Consumables are an efficient means to manage limited resources such as available memory, free space on a file system, network bandwidth or floating software licenses.

Consumables can be combined with default or user defined load parameters (see *sge_conf(5)* and *host_conf(5)*), i.e. load values can be reported for consumable attributes or the consumable flag can be set for load attributes. The Sun Grid Engine consumable resource management takes both the load (measuring availability of the resource) and the internal bookkeeping into account in this case, and makes sure that neither of both exceeds a given limit.

To enable consumable resource management the basic availability of a resource has to be defined. This can be done on a cluster global, per host and per queue basis while these categories may supersede each other in the given order (i.e. a host can restrict availability of a cluster resource and a queue can restrict host and cluster resources). The definition of resource availability is performed with the **complex_values** entry in *host_conf(5)* and *queue_conf(5)*. The **complex_values** definition of the “global” host specifies cluster global consumable settings. To each consumable complex attribute in a **complex_values** list a value is assigned which denotes the maximum available amount for that resource. The internal bookkeeping will subtract from this total the assumed resource consumption by all running jobs as expressed through the jobs’ resource requests.

Note – Jobs can be forced to request a resource and thus to specify their assumed consumption via the ‘force’ value of the requestable parameter (see above).

Note – A default resource consumption value can be pre-defined by the administrator for consumable attributes not explicitly requested by the job (see the default parameter below). This is meaningful only if requesting the attribute is not enforced as explained above.

See the *Sun Grid Engine Installation and Administration Guide* for examples on the usage of the consumable resources facility.

default

Meaningful only for consumable complex attributes (see **consumable** parameter above). Sun Grid Engine assumes the resource amount denoted in the **default** parameter implicitly to be consumed by jobs being dispatched to a host or queue managing the consumable attribute. Jobs explicitly requesting the attribute via the *-l* option to *qsub(1)* override this default value.

SEE ALSO

sge_intro(1), *qconf(1)*, *qsub(1)*, *uptime(1)*, *host_conf(5)*, *queue_conf(5)*, *cod_execd(8)*, *cod_qmaster(8)*, *cod_schedd(8)*, *Sun Grid Engine Installation and Administration Guide*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

host_conf – Sun Grid Engine execution host configuration file format

DESCRIPTION

Host_conf reflects the format of the template file for the execution host configuration. Via the **-ae** and **-me** options of the *qconf(1)* command, you can add execution hosts and modify the configuration of any execution host in the cluster. Default execution host entries are added automatically as soon as a *cod_execd(8)* registers to *cod_qmaster(8)* for the very first time from a certain host. The *qconf(1)* **-sel** switch can be used to display a list of execution host being currently configured in your Sun Grid Engine system. Via the **-se** option you can print the execution host configuration of a specified host.

The special hostname “global” can be used to define cluster global characteristics.

FORMAT

The format of a *host_conf* file is defined as follows:

hostname

The name of the execution host.

load_scaling

A comma separated list of scaling values to be applied to each or part of the load values being reported by the *cod_execd(8)* on the host and being defined in the cluster global “host” complex (see *complex(5)*). The load scaling factors are intended to level hardware or operating system specific differences between execution hosts. If, for example, the load average value (load_avg in the “host” complex; see also *uptime(1)*) of a multiprocessor machine is to be compared with a single processor machine the load as reported by the single CPU host needs to be weighted up against the multiprocessor load (given the same CPU hardware) to be comparable. The load scaling factors are integers being multiplied with the reported load quantities to constitute weighted load values. Thus, following the example given above, the load value of the single processor machine needs to be multiplied by the number of processors of the single processor machine to become comparable.

The syntax of a load factor specification is as follows: First the name of the load value (as defined in the “host” complex) is given and, separated by an equal sign, the load scaling value is provided. No blanks are allowed in between the load_scaling value string.

The parameter **load_scaling** is not meaningful for the definition of the “global” host.

complex_list

The comma separated list of administrator defined complexes (see *complex(5)* for details) to be associated with the host. Only complex attributes contained in the enlisted complexes and those from the “global” and “host” complex, which are implicitly attached to each host, can be used in the **complex_values** list below. In case of the “global” host, the “host” complex is not attached and only “global” complex attributes are allowed per default in the **complex_values** list of the “global” host.

The default value for this parameter is NONE, i.e. no administrator defined complexes are associated with the host.

complex_values

complex_values defines quotas for resource attributes managed via this host. Each complex attribute is followed by an “=” sign and the value specification compliant with the complex attribute type (see *complex(5)*). Quota specifications are separated by commas. Only attributes as defined in **complex_list** (see above) may be used.

The quotas are related to the resource consumption of all jobs on a host in the case of consumable resources (see *complex(5)* for details on consumable resources) or they are interpreted on a per job slot basis in the case of non-consumable resources. Consumable resource attributes are commonly used to manage free memory, free disk space or available floating software licenses while non-consumable attributes usually define distinctive characteristics like type of hardware installed.

For consumable resource attributes an available resource amount is determined by subtracting the current resource consumption of all running jobs on the host from the quota in the **complex_values** list. Jobs can only be dispatched to a host if no resource requests exceed any corresponding resource availability obtained by this scheme. The quota definition in the **complex_values** list is automatically replaced by the current load value reported for this attribute, if load is monitored for this resource and if the reported load value is more stringent than the quota. This effectively avoids oversubscription of resources.

Note – Load values replacing the quota specifications may have become more stringent because they have been scaled (see *load_scaling* above) and/or load adjusted (see *sched_conf(5)*). The *-F* option of *qstat(1)* and the load display in the *qmon(1)* queue control dialog (activated by clicking on a queue icon while the “Shift” key is pressed) provide detailed information on the actual availability of consumable resources and on the origin of the values taken into account currently.

Note – The resource consumption of running jobs (used for the availability calculation) as well as the resource requests of the jobs waiting to be dispatched either may be derived from explicit user requests during job submission (see the *-I* option to *qsub(1)*) or from a “default” value configured for an attribute by the administrator (see *complex(5)*). The *-r* option to *qstat(1)* can be used for retrieving full detail on the actual resource requests of all jobs in the system.

For non-consumable resources Sun Grid Engine simply compares the job's attribute requests with the corresponding specification in **complex_values** taking the relation operator of the complex attribute definition into account (see *complex(5)*). If the result of the comparison is "true", the host is suitable for the job with respect to the particular attribute. For parallel jobs each job slot to be occupied by a parallel task is meant to provide the same resource attribute value.

Note – Only numeric complex attributes can be defined as consumable resources and hence non-numeric attributes are always handled on a per job slot basis.

The default value for this parameter is NONE, i.e. no administrator defined resource attribute quotas are associated with the host.

load_values

This entry cannot be configured but is only displayed in case of a *qconf(1) -se* command. All load values are displayed as reported by the *cod_execd(8)* on the host. The load values are enlisted in a comma separated list. Each load value start with its name, followed by an equal sign and the reported value.

processors

This entry cannot be configured but is only displayed in case of a *qconf(1) -se* command. Its value is the number of processors which has been detected by *cod_execd(8)* on the corresponding host.

usage_scaling

This entry is only present in a Sun Grid Engine, Enterprise Edition system. It is not available in Sun Grid Engine.

The format is equivalent to **load_scaling** (see above), the only valid attributes to be scaled however are **cpu** for CPU time consumption, **mem** for Memory consumption aggregated over the life-time of jobs and **io** for data transferred via any I/O devices. The default NONE means "no scaling", i.e. all scaling factors are 1.

resource_capability_factor

This entry is only present in a Sun Grid Engine, Enterprise Edition system. It is not available in Sun Grid Engine.

The resource capability factor is used by Sun Grid Engine, Enterprise Edition when assigning jobs to execution hosts. The resource capability factor tells Sun Grid Engine, Enterprise Edition how the resources (CPU, memory, I/O, etc.) of one execution host compare to the resources of other execution hosts. This helps to ensure that a job requiring a large percentage of resources (i.e. lots of tickets) gets placed on an execution host containing a large percentage of the available resources. The load situation on the execution hosts is taken into account in addition, to guarantee that the selected host is both powerful enough and lightly loaded.

For example, you might consider setting your resource capability factors for each execution host based on the number of CPUs, the speed of the CPUs and the installed main memory:

$$\#_of_CPUs * (MHz/200) + GB_of_memory$$

This would give an execution host with 32 200 MHz CPUs and 10 gigabytes of memory a resource capability factor of 42, while an execution host with 24 200 MHz CPUs and 40 gigabytes of memory would get a resource capability factor of 64, i.e. memory has a significant impact in this example.

Other factors that you might want to consider in setting the resource capability factor are:

- | | |
|----------------------|----------------------------|
| job mix | - CPU or memory bound jobs |
| CPU benchmarks | - comparison by CPU vendor |
| megaflops (MFLOPS) | - for number crunching |
| I/O capabilities | - disk/network speed |
| available disk space | - at the execution host |

The resource capability factor is stored as a floating point double value. The range of values used is not important. Sun Grid Engine, Enterprise Edition only looks at the relation between values of different hosts.

SEE ALSO

sge_intro(1), *qconf(1)*, *uptime(1)*, *complex(5)*, *cod_execd(8)*, *cod_qmater(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

project – Sun Grid Engine, Enterprise Edition project entry file format

DESCRIPTION

The project object is only available in case of a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine has no project object.

Jobs can be submitted to projects in Sun Grid Engine, Enterprise Edition and a project can be assigned with a certain level of importance via the functional or the override policy. This level of importance is then inherited by the jobs executing under that project.

A list of currently configured projects can be displayed via the *qconf(1)* **–sprjl** option. The contents of each enlisted project definition can be shown via the **–sprj** switch. The output follows the *project* format description. New projects can be created and existing can be modified via the **–aprj**, **–mprj** and **–dprj** options to *qconf(1)*.

FORMAT

A project definition contains the following parameters:

name

The project name.

oticket

The amount of override tickets currently assigned to the project.

fshare

The current functional share of the project.

facl

A list of user access lists (ACLs - see *access_list(5)*) referring to those users being allowed to submit jobs to the project.

fxacl

A list of user access lists (ACLs - see *access_list(5)*) referring to those users being not allowed to submit jobs to the project.

SEE ALSO

sge_intro(1), *qconf(1)*, *access_list(5)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

qsi_conf – Sun Grid Engine Queuing System Interface (QSI) configuration file format

DESCRIPTION

qsi_conf defines the format of the QSI configuration file. The file is processed by the Queuing System Transfer Daemon (see *cod_qstd(8)*) and must reside in the corresponding spool directory on it's start-up.

Lines starting with a '#' or a ';' character are treated as comment lines. Empty lines are skipped. *Qsi_conf* requires the following entries to occur in the configuration file (the order of occurrence, however, is arbitrary):

queuing_system

The name of the queuing system to be interfaced by the host on which *cod_qstd(8)* processes this file. The name is arbitrary but must be unique.

transfer_queue

Attached Sun Grid Engine queue. Sun Grid Engine jobs being dispatched to this queue are transferred to the *cod_qstd(8)* maintaining this QSI configuration file.

submit

The calling sequence of a command procedure to submit a job passed by Sun Grid Engine to the queuing system to be interfaced. Invoked by *cod_qstd(8)*.

delete_job

The calling sequence of a command procedure to delete a job which has been passed by Sun Grid Engine to the queuing system to be interfaced. Invoked by *cod_qstd(8)* upon execution of the Sun Grid Engine *qdel(1)* command for that job.

suspend_queue

The calling sequence of a command procedure to suspend a job which has been passed by Sun Grid Engine to the queuing system to be interfaced. Invoked by *cod_qstd(8)* upon execution of the -s switch of the *qmod(1)* command for the corresponding transfer queue. Currently not implemented.

queuing_system_up

The calling sequence of a command procedure to poll for the foreign queuing system if it is up or not. Executed repeatedly by *cod_qstd(8)*.

job_status

The calling sequence of a command procedure to poll for the status of jobs which have been passed by Sun Grid Engine to the queuing system. Executed repeatedly by *cod_qstd(8)*.

job_finished

The calling sequence of a command procedure to be executed as soon as *cod_qstd(8)* recognizes the end of a job which has been passed by Sun Grid Engine to the queuing system to be interfaced. Usually used to clean up or save job specific data files.

load_sensor_command

The path to a command which is invoked by *cod_qstd(8)* periodically to retrieve load values from the foreign queuing system. The load sensor command is expected to follow the same rules as described for *cod_execd(8)* (see section **LOAD SENSORS**). If the same load parameters occur in both the load sensor command and the load sensor file (see below), the load sensor command values overwrite the values from the load sensor file. To be in effect, the reported load parameters need to be defined in the host complex (see *complex(5)*).

load_sensor_file

A file which contains fixed load values to be reported as the load of the foreign queuing system. Each line of the file is supposed to contain the name of the load parameter and then the associated value. If the same load parameters occur in both the load sensor command (see above) and the load sensor file, the load sensor command values overwrite the values from the load sensor file. To be in effect, the reported load parameters need to be defined in the host complex (see *complex(5)*).

RETURN VALUES AND OUTPUT HANDLING

The command procedures specified by the *qsi_conf* entries are supposed to behave on exit in a very specific way as defined below:

submit

On success the exit status should be 0 and the job-id should be returned to stdout. On failure the exit status should be 1 and an error message should be printed on stderr.

delete_job

On success the exit status should be 0. On failure the exit status should be 1 and an error message should be printed on stderr.

suspend_queue

On success the exit status should be 0. On failure the exit status should be 1 and an error message should be printed on stderr.

queuing_system_up

If the command succeeds, the exit status should be 0 and either “up” or “down” should be printed on stdout if the queuing system is in the corresponding state. The exit status should be 1 otherwise.

`job_status`

If the job is running the exit status should be 0 and the job status output should be printed to stdout. If the command fails, the exit status should be 1 and an error message should be printed on stderr. If the command succeeds, but the job is not running, the exit status should be 2.

`job_finished`

On success the exit status should be 0. On failure the exit status should be 1 and an error message should be printed on stderr.

`load_sensor_command`

Please refer to the description in section **LOAD SENSORS** of *cod_execd(8)*.

VARIABLES AVAILABLE IN CALLING SEQUENCES

The *qsi_conf* format allows for usage of a variety of variables in the calling sequences of the command procedures. The variables are expanded at runtime. The command procedure must process the variables as provided by the calling sequence definition and as expanded by *cod_qstd(8)* at runtime. Since some of the variables may not get a value after expansion, it is advised to quote such parameters with single quotes to ensure the number of arguments being passed to the command procedure being constant. You may alternatively want to add descriptive command line switches before variables in the calling sequence to simplify parameter parsing in the command procedures.

The following is a list of the available variables, the command procedures they are valid for and some additional remarks:

\$script_file

The job script file. Valid for the **submit** command procedure.

\$script_args

The arguments to the job scripts as provided by the *qsub(1)* command line. Valid for the **submit** command procedure. This variable should be quoted or prefixed with a switch as it may contain arbitrary (also 0) elements after expansion through *cod_qstd(8)*.

\$submitdir

The directory in which *qsub(1)* was executed (available only if -cwd switch to *qsub(1)* was present). Valid for the **submit** and the **job_finished** command procedure. This variable should be quoted or prefixed with a switch as it may be empty after expansion through *cod_qstd(8)*.

\$submithost

The host from which the job has been submitted. Valid for the **job_finished** command procedure.

\$owner

The owner who has submitted the job. Valid for the **delete_job** command procedure.

\$s_flag

The command interpreter (e.g. *sh(1)*, *csh(1)*) to execute the job script. Valid for the **submit** command procedure.

\$req_name

The request name of the job as defined by the *-N qsub(1)* switch. Valid for the **submit** and **job_finished** command procedure.

\$std_out

The name of the standard output redirection of the job as specified by the *qsub(1)* *-o* switch. This variable is not set if the switch is omitted and if Sun Grid Engine's default redirection file naming scheme is in effect. Valid for the **submit** and **job_finished** command procedure.

\$std_err

The name of the standard error redirection of the job as specified by the *qsub(1)* *-e* switch. This variable is not set if the switch is omitted and if Sun Grid Engine's default redirection file naming scheme is in effect. Valid for the **submit** and **job_finished** command procedure.

\$std_err_out

The name of the unified standard error/output redirection of the job as specified by the *qsub(1)* *-j y -o* option sequence. This variable is not set if the switch is omitted and if Sun Grid Engine's default redirection file naming scheme is in effect. Valid for the **submit** and **job_finished** command procedure.

\$qsub_args

The arguments to be passed to the queuing system as provided by the *-qs_args* switch in the *qsub(1)* command line. Valid for the **submit** command procedure. This variable should be quoted or prefixed with a switch as it may contain arbitrary (also 0) elements after expansion through *cod_qstd(8)*.

\$codine_job_id

The job-id as assigned to the job by Sun Grid Engine. Valid for the **submit**, the **delete_job** and the **job_finished** command procedure.

\$jobid

The job-id as assigned to the job by the foreign queuing system. Valid for the **job_status** command procedure.

\$architecture

The architecture-value of the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$queue_name

The queue name of the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$queue_hostname

The hostname of the transfer queue the job was scheduled to. This should be the same as the one *cod_qstd(8)* is running on. Valid for the **submit** command procedure.

\$at_time

The date and time at which the job is eligible for execution as specified by the *qsub(1)* -a switch. The date/time format conforms to the output of the *ctime(3)* or *asctime(3)* C-Library function (but does not contain the usual \n). The variable needs to be quoted, as it contains several space separated elements. If at-time was not specified this variable is set to the empty string. Valid for the **submit** command procedure.

\$s_cpu

The soft cpu time limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$h_cpu

The hard cpu time limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$s_fsize

The soft file size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$h_fsize

The hard file size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$s_data

The soft data segment size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$h_data

The hard data segment size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$s_stack

The soft stack segment size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$h_stack

The hard stack segment size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$s_core

The soft core file size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$h_core

The hard core file size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$s_rss

The soft resident set size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

\$h_rss

The soft resident set size limit as imposed on the job by the transfer queue the job was scheduled to. Valid for the **submit** command procedure.

RESTRICTIONS

The Sun Grid Engine Queuing System Interface must be licensed separately. Thus, this manual page is only applicable for installations using this feature.

FILES

<qsi_common_dir>/commands QSI configuration files

SEE ALSO

sgel_intro(1), *qsub(1)*, *complex(5)*, *cod_execd(8)*, *cod_qstd(8)*.

COPYRIGHT

See *sgel_intro(1)* for a full statement of rights and permissions.

NAME

qtask – file format of the qtask file.

DESCRIPTION

A *qtask* file defines which commands are submitted to Sun Grid Engine for remote execution by *qtcsh(1)*. The *qtask* file optionally may contain *qrsh(1)* command-line parameters. These parameters are passed to the *qrsh(1)* command being used by *qtcsh* to submit the commands.

A cluster global *qtask* file defining cluster wide defaults and a user specific *qtask* file eventually overriding and enhancing those definitions are supported. The cluster global file resides at `<cod_root>/<cell>/common/qtask`, while the user specific file can be found at `~/qtask`. An exclamation mark preceding command definitions in the cluster global can be used by the administrator to deny overriding of such commands by users.

FORMAT

The principle format of the *qtask* file is that of a tabulated list. Each line starting with a '#' character is a comment line. Each line despite comment lines defines a command to be started remotely.

Definition starts with the command name that must match exactly the name as typed in a *qtcsh(1)* command-line. Pathnames are not allowed in *qtask* files. Hence absolute or relative pathnames in *qtcsh(1)* command-lines always lead to local execution even if the commands itself are the same as defined in the *qtask* files.

The command name can be followed by an arbitrary number of *qrsh(1)* option arguments which are passed on to *qrsh(1)* by *qtcsh(1)*. An exclamation mark prefixing the command in the cluster global *qtask* file prevents overriding by the user supplied *qtask* file.

EXAMPLES

The following *qtask* file

```
netscape -l a=solaris64 -v DISPLAY=myhost:0
grep -l h=filesurfer
verilog -l veri_lic=1
```

designates the applications netscape, grep and verilog for interactive remote execution through Sun Grid Engine. Netscape is requested to run only on Solaris64 architectures with the **DISPLAY** environment variable set to 'myhost:0', grep only runs on the host named 'filesurfer' and verilog requests availability of a verilog license in order to get executed remotely.

SEE ALSO

sgc_intro(1), *qtcsh(1)*, *qrsh(1)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

queue_conf – Sun Grid Engine queue configuration file format

DESCRIPTION

Queue_conf reflects the format of the template file for the queue configuration. Via the **-aq** and **-mq** options of the *qconf(1)* command, you can add queues and modify the configuration of any queue in the cluster.

The *queue_conf* parameters take as values strings, integer decimal numbers or boolean, time and memory specifiers as well as comma separated lists. A time specifier either consists of a positive decimal, hexadecimal or octal integer constant, in which case the value is interpreted to be in seconds, or is built by 3 decimal integer numbers separated by colon signs where the first number counts the hours, the second the minutes and the third the seconds. If a number would be zero it can be left out but the separating colon must remain (e.g. 1:0:1 = 1::1 means 1 hours and 1 second).

Memory specifiers are positive decimal, hexadecimal or octal integer constants which may be followed by a multiplier letter. Valid multiplier letters are k, K, m and M, where k means multiply the value by 1000, K multiply by 1024, m multiply by 1000*1000 and M multiplies by 1024*1024. If no multiplier is present, the value is just counted in bytes.

FORMAT

The following list of *queue_conf* parameters specifies the *queue_conf* content:

qname

The name of the queue on the node (type string; template default: template).

hostname

The fully-qualified host name of the node (type string; template default: host.dom.dom.dom).

seq_no

With **sort_seq_no** (see *sched_conf(5)*) set to TRUE, this parameter specifies this queue's position in the scheduling order within the suitable queues for a job to be dispatched. It thus replaces the order by load policy that would rule otherwise.

Regardless of the **sort_seq_no** setting, *qstat(1)* reports queue information in the order defined by the value of the **seq_no**. Set this parameter to a monotonically increasing sequence. The type is number and the default is 0.

load_thresholds

load_thresholds is a list of load thresholds. Already if one of the thresholds is exceeded no further jobs will be scheduled to the queues on this node and *qmon(1)* will signal an overload condition for this node. Arbitrary load values being defined in the “host” and “global” complexes (see *complex(5)* for details) can be used.

The syntax is that of a comma separated list with each list element consisting of the name of a load value, an equal sign and the threshold value being intended to trigger the overload situation (e.g. `load.avg=175,users_logged_in=5`).

Note – Load values as well as consumable resources may be scaled differently for different hosts if specified in the corresponding execution host definitions (refer to *host_conf(5)* for more information). Load thresholds are compared against the scaled load and consumable values.

suspend_thresholds

A list of load thresholds with the same semantics as that of the **load_thresholds** parameter (see above) except that exceeding one of the denoted thresholds initiates suspension of one of multiple jobs in the queue. See the **nsuspend** parameter below for details on the number of jobs which are suspended.

nsuspend

The number of jobs which are suspended/enabled per time interval if at least one of the load thresholds in the **suspend_thresholds** list is exceeded or if no **suspend_threshold** is violated anymore respectively. **Nsuspend** jobs are suspended in each time interval until no **suspend_thresholds** are exceeded anymore or all jobs in the queue are suspended. Jobs are enabled in the corresponding way if the **suspend_thresholds** are no longer exceeded. The time interval in which the suspensions of the jobs occur is defined in **suspend_interval** below.

suspend_interval

The time interval in which further **nsuspend** jobs are suspended if one of the **suspend_thresholds** (see above for both) is exceeded by the current load on the host on which the queue is located. The time interval is also used when enabling the jobs.

migr_load_thresholds

A list of load thresholds with the same semantics as that of the **load_thresholds** parameter (see above) except that exceeding one of the denoted thresholds initiates migration of the jobs from the queue. This parameter has no effect in this release.

priority

The **priority** parameter specifies the *nice*(2) value at which jobs in this queue will be run. The type is number and the default is zero (which means no nice value is set explicitly).

max_migr_time

The time reserved for checkpointing jobs to be migrated and aborted. Checkpointing jobs due to be aborted are first sent a SIGTSTP. Everyone in the concerned process group may catch this signal and may react appropriately. After **max_migr_time** seconds, a SIGKILL is sent and the processes are aborted.

Note – If you set max_migr_time too high a user requesting full interactive usage may suffer max_migr_time seconds from a still running job. Max_migr_time is of type time and The default is 0 seconds.

migr_load_thresholds

A list of load thresholds with the same semantics as that of the **load_thresholds** parameter (see above) except that exceeding one of the denoted thresholds initiates migration of checkpointing jobs from the queue. It is recommended to set the migration load values high enough above the **load_thresholds** to prevent the jobs from forcing migrations by their own activity.

max_no_migr

The time a checkpointing job is allowed to spend in non-interruptible sections of the batch script. Non-interruptible sections are everything outside *qrestart*(1) commands. If a job exceeds this time limit it is killed and the job owner is notified. The default for **max_no_migr** is 2 minutes. It is of type time.

min_cpu_interval

The time between two automatic checkpoints in case of transparently checkpointing jobs. The maximum of the time requested by the user via *qsub*(1) and the time defined by the queue configuration is used as checkpoint interval.

Since checkpoint files may be considerably large and thus writing them to the file system may become expensive, users and administrators are advised to choose sufficiently large time intervals. **min_cpu_interval** is of type time and the default is 5 minutes (which usually is suitable for test purposes only).

processors

A set of processors in case of a multiprocessor execution host can be defined to which the jobs executing in this queue are bound. The value type of this parameter is a range description like that of the **-pe** option of *qsub(1)* (e.g. 1-4,8,10) denoting the processor numbers for the processor group to be used. Obviously the interpretation of these values relies on operating system specifics and is thus performed inside *cod_execd(8)* running on the queue host. Therefore, the parsing of the parameter has to be provided by the execution daemon and the parameter is only passed through *cod_qmaster(8)* as a string.

Currently, support is only provided for SGI multiprocessor machines running IRIX 6.2 and Digital UNIX multiprocessor machines. In the case of Digital UNIX only one job per processor set is allowed to execute at the same time, i.e. **slots** (see above) should be set to 1 for this queue.

qtype

The type of queue. Currently one of *batch*, *interactive*, *parallel* or *checkpointing* or any combination in a comma separated list. Alternatively, if the Sun Grid Engine Queuing System Interface (QSI) option is licensed, the type *transfer* can be specified to indicate a queue which passes jobs on to a foreign queuing system.

(type string; default: batch).

rerun

Defines a default behavior for jobs which are aborted by system crashes or manual “violent” (via *kill(1)*) shutdown of the complete Sun Grid Engine system (including the *cod_shepherd(8)* of the jobs and their process hierarchy) on the queue host. As soon as *cod_execd(8)* is restarted and detects that a job has been aborted for such reasons it can be restarted if the jobs are restartable. A job may not be restartable, for example, if it updates databases (first reads then writes to the same record of a database/file) because the abortion of the job may have left the database in an inconsistent state. If the owner of a job wants to overrule the default behavior for the jobs in the queue the **-r** option of *qsub(1)* can be used.

The type of this parameter is boolean, thus either TRUE or FALSE can be specified. The default is FALSE, i.e. do not restart jobs automatically.

slots

The maximum number of concurrently executing jobs allowed in the queue. Type is number.

tmpdir

The **tmpdir** parameter specifies the absolute path to the base of the temporary directory filesystem. When *cod_execd(8)* launches a job, it creates a uniquely-named directory in this filesystem for the purpose of holding scratch files during job execution. At job completion, this directory and its contents are removed automatically. The environment variables TMPDIR and TMP are set to the path of each jobs scratch directory (type string; default: /tmp).

shell

If either *posix_compliant* or *script_from_stdin* is specified as the **shell_start_mode** parameter in *sge_conf(5)* the **shell** parameter specifies the executable path of the command interpreter (e.g. *sh(1)* or *csh(1)*) to be used to process the job scripts executed in the queue. The definition of **shell** can be overruled by the job owner via the *qsub(1)* **-S** option.

The type of the parameter is string. The default is /bin/csh.

shell_start_mode

This parameter defines the mechanisms which are used to actually invoke the job scripts on the execution hosts. The following values are recognized:

unix_behavior

If a user starts a job shell script under UNIX interactively by invoking it just with the script name the operating system's executable loader uses the information provided in a comment such as `#!/bin/csh` in the first line of the script to detect which command interpreter to start to interpret the script. This mechanism is used by Sun Grid Engine when starting jobs if *unix_behavior* is defined as **shell_start_mode**.

posix_compliant

POSIX does not consider first script line comments such as `#!/bin/csh` as being significant. The POSIX standard for batch queuing systems (P1003.2d) therefore requires a compliant queuing system to ignore such lines but to use user specified or configured default command interpreters instead. Thus, if **shell_start_mode** is set to *posix_compliant* Sun Grid Engine will either use the command interpreter indicated by the **-S** option of the *qsub(1)* command or the **shell** parameter of the queue to be used (see above).

script_from_stdin

Setting the **shell_start_mode** parameter either to *posix_compliant* or *unix_behavior* requires you to set the umask in use for *cod_execd(8)* such that every user has read access to the active_jobs directory in the spool directory of the corresponding execution daemon. In case you have **prolog** and **epilog** scripts configured, they also need to be readable by any user who may execute jobs.

If this violates your site's security policies you may want to set **shell_start_mode** to *script_from_stdin*. This will force Sun Grid Engine to open the job script as well as the epilogue and prologue scripts for reading into STDIN as root (if *cod_execd(8)* was started as root) before changing to the job owner's user account. The script is then fed into the STDIN stream of the command interpreter indicated by the **-S** option of the *qsub(1)* command or the **shell** parameter of the queue to be used (see above).

Thus setting **shell_start_mode** to *script_from_stdin* also implies *posix_compliant* behavior.

Note – Feeding scripts into the STDIN stream of a command interpreter may cause trouble if commands like *rsh(1)* are invoked inside a job script as they also process the STDIN stream of the command interpreter. These problems can usually be resolved by redirecting the STDIN channel of those commands to come from */dev/null* (e.g. *rsh host date < /dev/null*).

Note – Any command-line options associated with the job are passed to the executing shell. The shell will only forward them to the job if they are not recognized as valid shell options.

The default for **shell_start_mode** is *posix_compliant*.

klog

The executable path of the klog utility on the queue host. It is used for AFS reauthentication. The type of the parameter is string; the default is */usr/local/bin/klog*.

Not supported in this release.

prolog

The executable path of a shell script that is started before execution of Sun Grid Engine jobs with the same environment setting as that for the Sun Grid Engine jobs to be started afterwards. An optional prefix "user@" specifies the user under which this procedure is to be started. This procedure is intended as a means for

the Sun Grid Engine administrator to automate the execution of general site specific tasks like the preparation of temporary file systems with the need for the same context information as the job. This queue configuration entry overwrites cluster global or execution host specific **prolog** definitions (see *sge_conf(5)*).

Note – prolog is executed *exactly* as the job script. Therefore, all implications described under the parameters `shell_start_mode` and `login_shells` below apply.

The default for **prolog** is the special value `NONE`, which prevents from execution of a prologue script. The special variables for constituting a command line are the same like in **prolog** definitions of the cluster configuration (see *sge_conf(5)*).

epilog

The executable path of a shell script that is started after execution of Sun Grid Engine jobs with the same environment setting as that for the Sun Grid Engine jobs that has just completed. An optional prefix “user@” specifies the user under which this procedure is to be started. This procedure is intended as a means for the Sun Grid Engine administrator to automate the execution of general site specific tasks like the cleaning up of temporary file systems with the need for the same context information as the job. This queue configuration entry overwrites cluster global or execution host specific **epilog** definitions (see *sge_conf(5)*).

Note – epilog is executed *exactly* as the job script. Therefore, all implications described under the parameters `shell_start_mode` and `login_shells` below apply.

The default for **epilog** is the special value `NONE`, which prevents from execution of an epilogue script. The special variables for constituting a command line are the same like in **prolog** definitions of the cluster configuration (see *sge_conf(5)*).

starter_method

The executable path given here is intended to be used as a starter facility which is responsible for starting the job itself.

Not supported in this release.

suspend_method

resume_method

terminate_method

These parameters can be used for overwriting the default method used by Sun Grid Engine for suspension, release of a suspension and for termination of a job. Per default, the signals SIGSTOP, SIGCONT and SIGKILL are delivered to the job to perform these actions. However, for some applications this is not appropriate.

If no executable path is given, Sun Grid Engine takes the specified parameter entries as the signal to be delivered instead of the default signal. A signal must be either a positive number or a signal name with “**SIG**” as prefix and the signal name as printed by *kill -l* (e.g. SIGTERM).

If an executable path is given (it must be an *absolute path* starting with a “/”) then this command together with its arguments is started by Sun Grid Engine to perform the appropriate action. The following special variables are expanded at runtime and can be used (besides any other strings which have to be interpreted by the procedures) to constitute a command line:

\$host

The name of the host on which the procedure is started.

\$job_owner

The user name of the job owner.

\$job_id

Sun Grid Engine’s unique job identification number.

\$job_name

The name of the job.

\$queue

The name of the queue.

\$job_pid

The pid of the job.

reauth_time

The time gap between consecutive AFS reauthentications. **Reauth_time** should be less than the ticket expiration time that is configured for the local AFS installation. The type of the parameter is time and the default value is 1 hour and 40 minutes, i.e. 100 minutes.

Not supported in this release.

notify

The time waited between delivery of SIGUSR1/SIGUSR2 notification signals and suspend/kill signals if job was submitted with the *qsub(1) -notify* option.

owner_list

The **owner_list** names the login names (in a comma separated list) of those users who are authorized to suspend this queue (Sun Grid Engine operators and managers can suspend queues by default). It is customary to set this field for queues on interactive workstations where the computing resources are shared between interactive sessions and Sun Grid Engine jobs, allowing the workstation owner to have priority access (type string; default: NONE).

user_lists

The **user_lists** parameter contains a comma separated list of so called user access lists as described in *access_list(5)*. Each user contained in at least one of the enlisted access lists has access to the queue. If the **user_lists** parameter is set to NONE (the default) any user has access being not explicitly excluded via the **xuser_lists** parameter described below. If a user is contained both in an access list enlisted in **xuser_lists** and **user_lists** the user is denied access to the queue.

xuser_lists

The **xuser_lists** parameter contains a comma separated list of so called user access lists as described in *access_list(5)*. Each user contained in at least one of the enlisted access lists is not allowed to access the queue. If the **xuser_lists** parameter is set to NONE (the default) any user has access. If a user is contained both in an access list enlisted in **xuser_lists** and **user_lists** the user is denied access to the queue.

projects

The **projects** parameter contains a comma separated list of projects that have access to the queue. Any projects not in this list are denied access to the queue. If set to NONE (the default), any project has access that is not specifically excluded via the **xprojects** parameter described below. If a project is in both the **projects** and **xprojects** parameters, the project is denied access to the queue. This parameter is only available in a Sun Grid Engine, Enterprise Edition system.

xprojects

The **xprojects** parameter contains a comma separated list of projects that are denied access to the queue. If set to NONE (the default), no projects are denied access other than those denied access based on the **projects** parameter described

above. If a project is in both the **projects** and **xprojects** parameters, the project is denied access to the queue. This parameter is only available in a Sun Grid Engine, Enterprise Edition system.

subordinate_list

A list of Sun Grid Engine queues, residing on the same host as the configured queue, to suspend when a specified count of jobs is running in this queue. The list specification is the same as that of the **load_thresholds** parameter above, e.g. `low_pri_q=5,small_q`. The numbers denote the job slots of the queue that have to be filled to trigger the suspension of the subordinated queue. If no value is assigned a suspension is triggered if all slots of the queue are filled.

On nodes which host more than one queue, you might wish to accord better service to certain classes of jobs (e.g., queues that are dedicated to parallel processing might need priority over low priority production queues; default: NONE).

complex_list

The comma separated list of administrator defined complexes (see *complex(5)* for details) to be associated with the queue. Only complex attributes contained in the enlisted complexes and those from the “global”, “host” and “queue” complex, which are implicitly attached to each queue, can be used in the **complex_values** list below.

The default value for this parameter is NONE, i.e. no administrator defined complexes are associated with the queue.

complex_values

complex_values defines quotas for resource attributes managed via this queue. The allowed complex attributes to appear in **complex_values** are defined by **complex_list** (see above). The syntax is the same as for **load_thresholds** (see above). The quotas are related to the resource consumption of all jobs in a queue in the case of consumable resources (see *complex(5)* for details on consumable resources) or they are interpreted on a per queue slot (see **slots** above) basis in the case of non-consumable resources. Consumable resource attributes are commonly used to manage free memory, free disk space or available floating software licenses while non-consumable attributes usually define distinctive characteristics like type of hardware installed.

For consumable resource attributes an available resource amount is determined by subtracting the current resource consumption of all running jobs in the queue from the quota in the **complex_values** list. Jobs can only be dispatched to a queue if no resource requests exceed any corresponding resource availability obtained by this scheme. The quota definition in the **complex_values** list is automatically

replaced by the current load value reported for this attribute, if load is monitored for this resource and if the reported load value is more stringent than the quota. This effectively avoids oversubscription of resources.

Note – Load values replacing the quota specifications may have become more stringent because they have been scaled (see *host_conf(5)*) and/or load adjusted (see *sched_conf(5)*). The *-F* option of *qstat(1)* and the load display in the *qmon(1)* queue control dialog (activated by clicking on a queue icon while the “Shift” key is pressed) provide detailed information on the actual availability of consumable resources and on the origin of the values taken into account currently.

Note – The resource consumption of running jobs (used for the availability calculation) as well as the resource requests of the jobs waiting to be dispatched either may be derived from explicit user requests during job submission (see the *-l* option to *qsub(1)*) or from a “default” value configured for an attribute by the administrator (see *complex(5)*). The *-r* option to *qstat(1)* can be used for retrieving full detail on the actual resource requests of all jobs in the system.

For non-consumable resources Sun Grid Engine simply compares the job’s attribute requests with the corresponding specification in **complex_values** taking the relation operator of the complex attribute definition into account (see *complex(5)*). If the result of the comparison is “true”, the queue is suitable for the job with respect to the particular attribute. For parallel jobs each queue slot to be occupied by a parallel task is meant to provide the same resource attribute value.

Note – Only numeric complex attributes can be defined as consumable resources and hence non-numeric attributes are always handled on a per queue slot basis.

The default value for this parameter is NONE, i.e. no administrator defined resource attribute quotas are associated with the queue.

calendar

specifies the **calendar** to be valid for this queue or contains NONE (the default). A calendar defines the availability of a queue depending on time of day, week and year. Please refer to *calendar_conf(5)* for details on the Sun Grid Engine calendar facility.

Note – Jobs can request queues with a certain calendar model via a “*-l c=<cal_name>*” option to *qsub(1)*.

initial_state

defines an initial state for the queue either when adding the queue to the system for the first time or on start-up of the *cod_execd(8)* on the host on which the queue resides. Possible values are:

default

The queue is enabled when adding the queue or is reset to the previous status when *cod_execd(8)* comes up (this corresponds to the behavior in earlier Sun Grid Engine releases not supporting *initial_state*).

enabled

The queue is enabled in either case. This is equivalent to a manual and explicit '*qmod -e*' command (see *qmod(1)*).

disabled

The queue is disabled in either case. This is equivalent to a manual and explicit '*qmod -d*' command (see *qmod(1)*).

fshare

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The functional shares of the queue (i.e. job class). Jobs executing in this queue may get functional tickets derived from the relative importance of the queue if the functional policy is activated.

oticket

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The override tickets of the queue (i.e. job class). Sun Grid Engine, Enterprise Edition distributes the configured amount of override tickets among all jobs executing in this queue.

RESOURCE LIMITS

The first two resource limit parameters, **s_rt** and **h_rt**, are implemented by Sun Grid Engine. They define the “real time” or also called “elapsed” or “wall clock” time having passed since the start of the job. If **h_rt** is exceeded by a job running in the queue, it is aborted via the SIGKILL signal (see *kill(1)*). If **s_rt** is exceeded, the job is first “warned” via the SIGUSR1 signal (which can be caught by the job) and finally aborted after the notification time defined in the queue configuration parameter **notify** (see above) has passed.

The resource limit parameters **s_cpu** and **h_cpu** are implemented by Sun Grid Engine as a job limit. They impose a limit on the amount of combined CPU time consumed by all the processes in the job. If **h_cpu** is exceeded by a job running in the queue, it is aborted via a SIGKILL signal (see *kill(1)*). If **s_cpu** is exceeded, the job is sent a SIGXCPU signal which can be caught by the job. If you wish to allow a job to be “warned” so it can exit gracefully before it is killed then you should set the **s_cpu** limit to a lower value than **h_cpu**. For parallel processes, the limit is applied per slot which means that the limit is multiplied by the number of slots being used by the job before being applied.

The resource limit parameters **s_vmem** and **h_vmem** are implemented by Sun Grid Engine as a job limit. They impose a limit on the amount of combined virtual memory consumed by all the processes in the job. If **h_vmem** is exceeded by a job running in the queue, it is aborted via a SIGKILL signal (see *kill(1)*). If **s_vmem** is exceeded, the job is sent a SIGXCPU signal which can be caught by the job. If you wish to allow a job to be “warned” so it can exit gracefully before it is killed then you should set the **s_vmem** limit to a lower value than **h_vmem**. For parallel processes, the limit is applied per slot which means that the limit is multiplied by the number of slots being used by the job before being applied.

The remaining parameters in the queue configuration template specify per job soft and hard resource limits as implemented by the *setrlimit(2)* system call. See this manual page on your system for more information. By default, each limit field is set to infinity (which means RLIM_INFINITY as described in the *setrlimit(2)* manual page). The value type for the CPU-time limits **s_cpu** and **h_cpu** is time. The value type for the other limits is memory.

Note – Not all systems support *setrlimit(2)*.

Note – **s_vmem** and **h_vmem** (virtual memory) are only available on systems supporting RLIMIT_VMEM (see *setrlimit(2)* on your operating system).

The UNICOS operating system supplied by SGI/Cray does not support the *setrlimit(2)* system call, using their own resource limit-setting system call instead. For UNICOS systems only, the following meanings apply:

s_cpu

The per-process CPU time limit in seconds.

s_core

The per-process maximum core file size in bytes.

s_data

The per-process maximum memory limit in bytes.

s_vmem

The same as **s_data** (if both are set the minimum is used).

h_cpu

The per-job CPU time limit in seconds.

`h_data`

The per-job maximum memory limit in bytes.

`h_vmem`

The same as `h_data` (if both are set the minimum is used).

`h_fsize`

The total number of disk blocks that this job can create.

SEE ALSO

sgc_intro(1), *cs(1)*, *qconf(1)*, *qmon(1)*, *qrestart(1)*, *qstat(1)*, *qsub(1)*, *sh(1)*, *nice(2)*, *setrlimit(2)*, *access_list(5)*, *calendar_conf(5)*, *sgc_conf(5)*, *complex(5)*, *host_conf(5)*, *sched_conf(5)*, *qsi_conf(5)*, *cod_execd(8)*, *cod_qmaster(8)*, *cod_qstd(8)*, *cod_shepherd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

`sched_conf` – Sun Grid Engine default scheduler configuration file

DESCRIPTION

sched_conf defines the configuration file format for Sun Grid Engine's default scheduler provided by *cod_schedd(8)*. In order to modify the configuration, use the graphical user's interface *qmon(1)* or the *-msconf* option of the *qconf(1)* command. A default configuration is provided together with the Sun Grid Engine distribution package.

FORMAT

The following parameters are recognized by the Sun Grid Engine scheduler if present in *sched_conf*:

algorithm

Allows for the selection of alternative scheduling algorithms.

Currently **default** is the only allowed setting.

load_formula

A simple algebraic expression used to derive a single weighted load value from all or part of the load parameters reported by *cod_exeecd(8)* for each host and from all or part of the consumable resources (see *complex(5)*) being maintained for each host. The load formula expression syntax is that of a summation weighted load values, that is:

$$\text{load_val1}[*w1][\{+|- \}]\text{load_val2}[*w2][\{+|- \}]\dots$$

Note – No blanks are allowed in the load formula.

The load values and consumable resources (`load_val1, ...`) are specified by the name defined in the complex (see *complex(5)*).

Note – Administrator defined load values (see the `load_sensor` parameter in `sge_conf(5)` for details) and consumable resources available for all hosts (see `complex(5)`) may be used as well as Sun Grid Engine default load parameters.

The weighting factors (`w1`, ...) are positive integers. After the expression is evaluated for each host the results are assigned to the hosts and are used to sort the hosts corresponding to the weighted load. The sorted host list is used to sort queues subsequently.

The default load formula is “`load_avg`”.

job_load_adjustments

The load, which is imposed by the Sun Grid Engine jobs running on a system varies in time, and often, e.g. for the CPU load, requires some amount of time to be reported in the appropriate quantity by the operating system. Consequently, if a job was started very recently, the reported load may not provide a sufficient representation of the load which is already imposed on that host by the job. The reported load will adapt to the real load over time, but the period of time, in which the reported load is too low, may already lead to an oversubscription of that host. Sun Grid Engine allows the administrator to specify **job_load_adjustments** which are used in the Sun Grid Engine scheduler to compensate for this problem.

The **job_load_adjustments** are specified as a comma separated list of arbitrary load parameters or consumable resources and (separated by an equal sign) an associated load correction value. Whenever a job is dispatched to a host by `cod_schedd(8)`, the load parameter and consumable value set of that host is increased by the values provided in the **job_load_adjustments** list. These correction values are decayed linearly over time until after **load_adjustment_decay_time** from the start the corrections reach the value 0. If the **job_load_adjustments** list is assigned the special denominator `NONE`, no load corrections are performed.

The adjusted load and consumable values are used to compute the combined and weighted load of the hosts with the **load_formula** (see above) and to compare the load and consumable values against the load threshold lists defined in the queue configurations (see `queue_conf(5)`). If your **load_formula** simply consists of the CPU load average parameter `load_avg` and if your jobs are very compute intensive, you might want to set the **job_load_adjustments** list to `load_avg=100`, which means that every new job dispatched to a host will require 100 % CPU time and thus the machine’s load is instantly raised by 100.

load_adjustment_decay_time

The load corrections in the “**job_load_adjustments**” list above are decayed linearly over time from the point of the job start, where the corresponding load or consumable parameter is raised by the full correction value, until after a time period of “**load_adjustment_decay_time**”, where the correction becomes 0. Proper values for “**load_adjustment_decay_time**” greatly depend upon the load or consumable parameters used and the specific operating system(s). Therefore, they can only be determined on-site and experimentally. For the default *load_avg* load parameter a “**load_adjustment_decay_time**” of 7 minutes has proven to yield reasonable results.

maxujobs

The maximum number of jobs any user may have running in a Sun Grid Engine cluster at the same time. If set to 0 (default) the users may run an arbitrary number of jobs. If the **user_sort** scheduling policy is active (see below) the scheduler allows at the most **maxujobs** in each priority group

The **maxujobs** parameter has no effect in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine, Enterprise Edition provides more sophisticated means to control share entitlement.

maxgjobs

Not implemented yet. Provided for later extension.

schedule_interval

At the time *cod_schedd(8)* initially registers to *cod_qmaster(8)* **schedule_interval** is used to set the time interval in which *cod_qmaster(8)* sends scheduling event updates to *cod_schedd(8)*. A scheduling event is a status change that has occurred within *cod_qmaster(8)* which may trigger or affect scheduler decisions (e.g. a job has finished and thus the allocated resources are available again).

In the Sun Grid Engine default scheduler the arrival of a scheduling event report triggers a scheduler run. The scheduler waits for event reports otherwise.

Schedule_interval is a time value (see *queue_conf(5)* for a definition of the syntax of time values).

user_sort

Sun Grid Engine usually schedules user jobs corresponding to a first-come-first-served policy. In case a user submits a large amount of jobs in very short time, this can lead to a rather unfair situation, because all users submitting afterwards are blocked until most of the first user’s jobs are completed. Therefore, Sun Grid

Engine allows to change this policy to the so called equal share sort: As soon as a user has a job running his other jobs are sorted to the end of the pending jobs list. Thus, the first jobs of all other users have comparable chances to find a queue.

Note – The equal share sort only applies within the same job priority category (refer to the *mp* option of the *qsub(1)* and *qalter(1)* commands for more information).

The default for **user_sort** is **FALSE**.

queue_sort_method

If this parameter is set to **seqno**, Sun Grid Engine will use the **seq_no** parameter as configured in the current queue configurations (see *queue_conf(5)*) as first criterion to produce a sorted queue list. The **load_formula** (see above) has only a meaning if two queues have equal sequence numbers. If **queue_sort_method** is set to **load** the load according the **load_formula** is the first criterion and the sequence number is only used if two hosts have the same load. The sequence number sorting is most useful if you want to define a fixed order in which queues are to be filled (e.g. the cheapest resource first).

The default for this parameter is **load**.

grd_schedule_interval

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The time period between job priority adjustments by the Sun Grid Engine, Enterprise Edition global dynamic scheduler (GDS). Valid values are specified of type time as specified in *queue_conf(5)*.

halftime

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

When executing under a share based policy, Sun Grid Engine, Enterprise Edition “ages” (i.e. decreases) usage to implement a sliding window for achieving the share entitlements as defined by the share tree. The **halftime** defines the time interval in which accumulated usage will have been decayed to half its original value. Valid values are specified of type time as specified in *queue_conf(5)*.

usage_weight_list

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

Sun Grid Engine, Enterprise Edition accounts for the consumption of the resources CPU-time, memory and IO to determine the usage which is imposed on a system by a job. A single usage value is computed from these three input parameters by multiplying the individual values by weights and adding them up. The weights are defined in the **usage_weight_list**. The format of the list is

`cpu=wcpu,mem=wmem,io=wio`

where wcpu, wmem and wio are the configurable weights. The weights are real number. The sum of all tree weights should be 1.

compensation_factor

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

Determines how fast Sun Grid Engine, Enterprise Edition should compensate for past usage below of above the share entitlement defined in the share tree. Recommended values are between 2 and 10, where 10 means faster compensation.

weight_user

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The relative importance of the user shares in the functional policy. Values are of type real.

weight_project

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The relative importance of the project shares in the functional policy. Values are of type real.

weight_jobclass

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The relative importance of the job class (i.e. queue) shares in the functional policy. Values are of type real.

weight_department

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The relative importance of the department shares in the functional policy. Values are of type real.

weight_job

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The relative importance of the job shares in the functional policy. Values are of type real.

weight_tickets_functional

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The maximum number of functional tickets available for distribution by Sun Grid Engine, Enterprise Edition. Determines the relative importance of the functional policy.

weight_tickets_share

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The maximum number of share based tickets available for distribution by Sun Grid Engine, Enterprise Edition. Determines the relative importance of the share tree policy.

weight_deadline

This parameter is only available in a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine does not support this parameter.

The maximum number of deadline tickets available for distribution by Sun Grid Engine, Enterprise Edition. Determines the relative importance of the deadline policy.

schedd_job_info

The default scheduler can keep track why jobs could not be scheduled during the last scheduler run. This parameter enables or disables the observation. The value **true** enables the monitoring **false** turns it off.

It is also possible to activate the observation only for certain jobs. This will be done if the parameter is set to **job_list** followed by a comma separated list of job ids.

The user can obtain the collected information with the command `qstat -j`.

FILES

`<codine_root>/<cell>/common/sched_configuration`
cod_schedd configuration

SEE ALSO

sgc_intro(1), qalter(1), qconf(1), qstat(1), qsub(1), complex(5), queue_conf(5), cod_execd(8), cod_qmaster(8), cod_schedd(8), Sun Grid Engine Installation and Administration Guide.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

SHARE_TREE(5)

NAME

share_tree – Sun Grid Engine, Enterprise Edition share tree file format

DESCRIPTION

The share tree object is only available in case of a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine has no share tree object.

The share tree defines the long-term resource entitlements of users/projects and of a hierarchy of arbitrary groups thereof.

The current share tree can be displayed via the *qconf(1)* **-sstree** option. The output follows the *share_tree* format description. A share tree can be created and an existing can be modified via the **-astree** and **-mstree** options to *qconf(1)*. Individual share tree nodes can be created, modified, deleted, or shown via the **-astnode**, **-dstnode**, **-mstnode**, and **-sstnode** options to *qconf(1)*.

FORMAT

The format of a share tree file is defined as follows:

- ❑ A new node starts with the attribute **id**, and equal sign and the numeric identification number of the node. Further attributes of that node follow until another **id**-keyword is encountered.
- ❑ The attribute **childnodes** contains a comma separated list of child nodes to this node.
- ❑ The parameter **name** refers to an arbitrary name for the node or to a corresponding user (see *user(5)*) or project (see *project(5)*) if the node is a leaf node of the share tree. The name for the root node of the tree is “Root” by convention.
- ❑ The parameter **shares** defines the share of the node among the nodes with the same parent node.

SEE ALSO

sge_intro(1), *qconf(1)*, *project(5)*, *user(5)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

user – Sun Grid Engine, Enterprise Edition user entry file format

DESCRIPTION

The user object is only available in case of a Sun Grid Engine, Enterprise Edition system. Sun Grid Engine has no user object.

A user entry is used in Sun Grid Engine, Enterprise Edition to store ticket and usage information on a per user basis. Maintaining user entries for all users participating in a Sun Grid Engine, Enterprise Edition system is required if Sun Grid Engine, Enterprise Edition is operated under a user share tree policy.

A list of currently configured user entries can be displayed via the *qconf(1)* **-suserl** option. The contents of each enlisted user entry can be shown via the **-suser** switch. The output follows the *user* format description. New user entries can be created and existing can be modified via the **-auser**, **-muser** and **-duser** options to *qconf(1)*.

FORMAT

A user entry contains four parameters:

name

The user name.

oticket

The amount of override tickets currently assigned to the user.

fshare

The current functional share of the user.

default_project

The default project of the user.

SEE ALSO

sgc_intro(1), *qconf(1)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

`cod_commd` – Sun Grid Engine communication agent

SYNOPSIS

```
cod_commd [ -S ] [ -a aliasfile ] [ -dhr ] [ -help ]  
          [ -ll loglevel ] [ -ml fname ] [ -nd ] [ -p port ]  
          [ -s service ]
```

DESCRIPTION

All network communication in a Sun Grid Engine cluster is performed via the communication daemons *cod_commd*. Client programs like *qsub(1)* or *qstat(1)* as well as Sun Grid Engine daemons such as *cod_qmaster(8)* or *cod_execd(8)* use the service provided by *cod_commd* in order to send/receive messages to/from other Sun Grid Engine components.

cod_commd handles an arbitrary number of concurrent synchronous or asynchronous communications. Usually one *cod_commd* is started up automatically on each host on which *cod_qmaster(8)*, *cod_execd(8)* or/and *cod_schedd(8)* are invoked. It is however possible to connect multiple hosts via one *cod_commd* or to use a *cod_commd* on a submit or administrative Sun Grid Engine host (without running one of the other Sun Grid Engine daemons) as communication agent for the Sun Grid Engine client programs invoked from that host.

OPTIONS

-S

Forces secure ports to be used for communication between *cod_commds* and between other Sun Grid Engine components and the *cod_commds*. This requires all Sun Grid Engine daemons to be started with root permission and the client programs to be configured set-uid root. In turn, it ensures that unauthorized communication is prohibited for non-root accounts.

-a aliasfile

A file containing Sun Grid Engine host aliases used by the *cod_commd* to resolve Sun Grid Engine unique hostnames for all hosts in the cluster. The hostname resolving service of *cod_commd* is also used by all other Sun Grid Engine components. The file format and the implication of its usage are described in *sge_h_aliases(5)*.

-dhr

The hostname resolving C-library functions (such as *gethostent(3)*, *gethostbyname(3)* and *gethostbyaddr(3)*) perform some kind of caching on some OS architectures. Network wide hostname databases distributed by services such as **DNS** (**D**omain **N**ame **S**ervice) and **NIS** (**N**etwork **I**nformation **S**ervice) are updated with a delay of several minutes. This only affects applications which repeatedly resolve hostnames (such as *cod_commd*). At start-up of a program the most recent information is accessed, thus commands like *telnet(1)* or *nslookup(1)* are not affected.

However, for *cod_commd* it makes no sense to resolve hostnames anytime (the returned information may be out of date anyway) and resolving can be an expensive operation if the network is overloaded and/or **NIS** or **DNS** servers are very busy. Therefore, *cod_commd* resolves hostname information from time to time only.

Yet, if hostname resolving still causes problems due to network load, for example, it can be turned off with the **-dhr** switch. The administrator has to be aware, that if the hostname resolving is turned off, *cod_commd* has to be restarted as soon as the hostname databases change significantly.

-help

Prints a listing of all options.

-ll loglevel

Sets a logging level for error tracing. The error trace information is written to the file */tmp/commd/err.<pid>*. However, the directory */tmp/commd* must be present, otherwise the tracing output is discarded. At present, 255 is the only valid logging level.

-nd

Do not daemonize. If started with **-nd**, *cod_commd* maintains its connection to the controlling terminal and thus outputs trace information directly to the terminal from which *cod_commd* was invoked. The trace information is the same as being accessible via the **-ll** option (see above).

-p port_number

Use this TCP port for communication with other commds.

-s service_name

Use this service name and thus the associated TCP port for communication with other commds.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

(Does not affect the behavior of *cod_commd* but of the other Sun Grid Engine components contacting *cod_commd*.) If set, specifies the host on which the particular *cod_commd* to be used for Sun Grid Engine communication of arbitrary Sun Grid Engine client programs or daemons resides. Per default the local host is used.

RESTRICTIONS

cod_commd usually is invoked by a starting *cod_qmaster(8)* and *cod_execd(8)* and thus is running under root permission. If started by a normal user the **-S** switch may not be used as the secure mode requires root permission (see above).

SEE ALSO

sge_intro(1), *sge_h_aliases(5)*, *cod_execd(8)*, *cod_qmaster(8)*, *commdctl(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`cod_execd` – Sun Grid Engine job execution agent

SYNOPSIS

`cod_execd` [**-help**] [**-lj** `log_file`] [**-nostart-commnd**]

DESCRIPTION

`cod_execd` controls the Sun Grid Engine queues local to the machine `cod_execd` is running on and executes/controls the jobs sent from `cod_qmaster(8)` to be run on these queues.

Together with `cod_execd` a `cod_commd(8)` is brought up automatically on the same machine (if not already present).

OPTIONS

-help

Prints a listing of all options.

-lj `log_file`

Enables job logging. All actions taken by `cod_execd` from receiving the job until returning it to `cod_qmaster(8)` are logged to the `log_file`. This feature is also available with the `cod_qmaster(8)` daemon.

-nostart-commnd

Do not start up `cod_commd(8)` automatically with `cod_execd` and evaluate the `COMMD_HOST` environment variable to find the corresponding `cod_commd(8)`.

LOAD SENSORS

If a **load sensor** is configured for *cod_execd* via either the global or the execution host specific cluster configuration (see *sge_conf(5)*) the executable path of the load sensor is invoked by *cod_execd* on a regular basis and delivers one or multiple load figures for the execution host (e.g. users currently logged in) or on the complete cluster (e.g. free disk space on a network wide scratch file system). The load sensor may be a script or a binary executable. In either case its handling of the **STDIN** and **STDOUT** stream and its control flow must comply to the following rules:

The load sensor has to be written as infinite loop waiting at a certain point for input from **STDIN**. If the string **quit** is read from **STDIN**, the load sensor is supposed to exit. As soon as an end-of-line is read from **STDIN** a load data retrieval cycle is supposed to start. The load sensor then performs whatever operation is necessary to compute the desired load figures. At the end of the cycle the load sensor writes the result to **stdout**. The format is as follows:

- ❑ A load value report starts with a line containing nothing but the word **start**.
- ❑ Individual load values are separated by newlines.
- ❑ Each load value report consists of three parts separated by colons (":") and containing no blanks.
- ❑ The first part of a load value information is either the name of the host for which load is reported or the special name "global".
- ❑ The second part is the symbolic name of the load value as defined in the host or global complex list (see *complex(5)* for details). If a load value is reported for which no entry in the host or global complex list exists, the reported load value is not used.
- ❑ The third part is the measured load value.
- ❑ A load value report ends with a line with the word **end**.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of */usr/CODINE* is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *cod_execd* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to **stderr**. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *cod_execd* client resides. Only evaluated if the **-nostart-commd** option was specified at the *cod_execd* command-line. Per default the local host is used.

RESTRICTIONS

cod_execd usually is started from root on each machine in the Sun Grid Engine pool. If started by a normal user, a spool directory must be used to which the user has read/write access. In this case only jobs being submitted by that same user are treated correctly by the system.

FILES

<codine_root>/<cell>/common/configuration
 Sun Grid Engine global configuration
<codine_root>/<cell>/common/local_conf/<host>
 Sun Grid Engine host specific configuration
<codine_root>/<cell>/spool/<host>
 Default execution host spool directory
<codine_root>/<cell>/common/act_qmaster
 Sun Grid Engine master host file

SEE ALSO

sge_intro(1), *sge_conf(5)*, *complex(5)*, *cod_commd(8)*, *cod_qmaster(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`cod_qmaster` – Sun Grid Engine master control daemon

SYNOPSIS

```
cod_qmaster [ -help ] [ -lj log_file ] [ -nohist ]  
          [ -noread-argfile ] [ -nostart-commd ]  
          [ -nostart-schedd ] [ -nowrite-argfile ] [ -s ]  
          [ -truncate-argfile ]
```

cod_qmaster -show-license

DESCRIPTION

cod_qmaster controls the overall Sun Grid Engine behavior in a cluster. For the purpose of scheduling jobs *cod_qmaster* cooperates with *cod_schedd(8)*. At start-up of *cod_qmaster* *cod_commd(8)* is usually brought up automatically on the same machine (if not already present).

OPTIONS

-help

Prints a listing of all options.

-lj *log_file*

Enables job logging. All actions taken by *cod_qmaster* from submit to job exit are logged to the *log_file*. This feature is also available with the *cod_execd(8)* daemon.

-nohist

During usual operation *cod_qmaster* dumps a history of queue, complex and host configuration changes to a history database. This database is primarily used with the *qacct(1)* command to allow for *qsub(1)* like *-l* resource requests in the *qacct(1)* command-line. This switch suppresses writing to this database.

-noread-argfile

On primary start-up, *cod_qmaster* writes its command-line arguments to a file. During later start-ups, this argument file will be read and the options contained in the file will be used as if supplied at the command-line. This option suppresses reading of the argument file.

–nostart-commnd

Do not start-up *cod_commd(8)* automatically with *cod_qmaster*.

–nostart-schedd

Do not startup *cod_schedd(8)* automatically with *cod_qmaster*. *cod_qmaster* currently does not start *cod_schedd(8)* automatically. Thus this option has no effect.

–nowrite-argfile

On primary start-up, *cod_qmaster* writes its command-line arguments to a file. During later start-ups, this argument file will be read and the options contained in the file will be used as if supplied at the command-line. This option suppresses writing of the argument file.

–s

turns on *cod_qmasters* silent mode. Usually *cod_qmaster* displays a license information on startup and waits for a return to continue. With the -s switch *cod_qmaster* starts up silently.

–show-license

Displays the current licensing information for your Sun Grid Engine system. This option also works if your license has expired and *cod_qmaster* would exit immediately otherwise. Use the displayed information to request an appropriate license from your Sun Grid Engine support contact.

–truncate-argfile

On primary start-up, *cod_qmaster* writes its command-line arguments to a file. During later start-ups, this argument file will be read and the options contained in the file will be used as if supplied at the command-line. This option truncates the argument file.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of /usr/CODINE is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *cod_qmaster* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

RESTRICTIONS

cod_qmaster is usually started from root on the master or shadow master machines of the cluster (refer to the *Sun Grid Engine Installation and Administration Guide* for more information about the configuration of shadow master hosts). If started by a normal user, a master spool directory must be used to which the user has read/write access. In this case only jobs being submitted by that same user are treated correctly by the system.

FILES

<codine_root>/<cell>/common/configuration
Sun Grid Engine global configuration
<codine_root>/<cell>/common/local_conf/<host>
Sun Grid Engine host specific configuration
<codine_root>/<cell>/common/history
History database
<codine_root>/<cell>/common/qmaster_args
cod_qmaster argument file
<codine_root>/<cell>/spool
Default master spool directory

SEE ALSO

sge_intro(1), *sge_conf(5)*, *cod_commd(8)*, *cod_execd(8)*, *cod_schedd(8)*, *cod_shadowd(8)*, *Sun Grid Engine Installation and Administration Guide*

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`cod_qstd` – Sun Grid Engine foreign queueing system interface daemon

SYNOPSIS

`cod_qstd` [**-help**] [**-nostart-commd**]

DESCRIPTION

cod_qstd provides an interface between Sun Grid Engine and other queueing systems being accessed via so called transfer queues.

Together with *cod_execd* a *cod_commd(8)* is brought up automatically on the same machine (if not already present).

For information on how *cod_qstd* can be configured see the section FILES below.

OPTIONS

-help

Prints a listing of all options.

-nostart-commd

Do not start up *cod_commd(8)* automatically with *cod_qstd* and evaluate the **COMMD_HOST** environment variable to locate the corresponding *cod_commd(8)*.

RESTRICTIONS

cod_qstd may only be started from root. If started by a normal user, a spool directory must be used to which the user has read/write access. In this case only jobs being submitted by that same user are treated correctly by the system.

The Sun Grid Engine Queueing System Interface must be licensed separately. Thus, this manual page is only applicable for installations using this feature.

ENVIRONMENTAL VARIABLES

CODINE_ROOT

If not set, a default of /usr/CODINE is used. In either case, the spool directory path is set to *<codine_root>/<cell>/spool/qstd/unqualified_hostname*. This setting may be overwritten by the **-s** command line option (see above).

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *cod_qstd* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *cod_qstd* client resides. Only evaluated if the **-nostart-commd** option was specified at the *cod_qstd* command-line. Per default the local host is used.

FILES

The configuration for the QSI defining how *cod_qstd* is supposed to interface the foreign queuing systems is expected in a so called QSI common directory containing the following files:

commands*

Every file with a name starting with the string **commands** is considered to contain the configuration for a foreign queueing system interface. Please refer to *qsi_conf(5)* for a detailed description of the file format. At least one such file must exist before *cod_qstd* is started up.

The location of the QSI common directory is defined by the cluster configuration parameter **qsi_common_dir** (see *sge_conf(5)*).

The *cod_qstd* spool directory contains several files, most of them used to temporarily store information. Two of the files are important with respect to trouble shooting:

messages

The system messages and error logging file of *cod_qstd*.

log_of_commands

This file contains log-entries for each queueing system command procedure invoked by *cod_qstd*.

The *cod_qstd* spool directory is a sub-directory named **qsi** to the *cod_execd(8)* spool directory of the corresponding execution hosts.

In addition, the following files and directory are relevant to *cod_qstd*.

<codine_root>/<cell>/common/qsi

Default *cod_qstd* configuration

<codine_root>/<cell>/spool/<host>/qsi

Default *cod_qstd* spool directory

<codine_root>/<cell>/common/act_qmaster

cod_qmaster name file

SEE ALSO

sge_intro(1), *qsi_conf(5)*, *cod_commd(8)*, *cod_qmaster(8)*, *Sun Grid Engine Installation and Administration Guide*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`cod_schedd` – Sun Grid Engine job scheduling agent

SYNOPSIS

`cod_schedd` [**-help**] [**-k**] [**-salg**]

DESCRIPTION

`cod_schedd` computes the scheduling decision in a Sun Grid Engine cluster. The information necessary for the decision is retrieved from `cod_qmaster(8)` via the Sun Grid Engine Application Programmers Interface (API - see `cod_api(3)` for details). After applying the scheduling algorithm, `cod_schedd` communicates the scheduling decision back to `cod_qmaster(8)` again via the Sun Grid Engine API. In order to trigger a `cod_schedd` run, `cod_qmaster(8)` samples changes in the cluster status and notifies `cod_schedd` in periodical time intervals.

Together with `cod_schedd` a `cod_commd(8)` is brought up automatically on the same machine (if not already present).

By using the `-tsm` option of the `qconf(1)` command, `cod_schedd` can be forced to print trace messages of its next scheduling run to the file `<codine_root>/<cell>/common/schedd_runlog`. The messages indicate the reasons for jobs and queues not being selected in that run

OPTIONS

-help

Prints a listing of all options.

-k

Initiates a controlled shutdown of a running `cod_schedd` on the same host.

-salg

Display a list of feasible scheduling algorithms to choose from via the scheduler configuration (see `sched_conf(5)`).

ENVIRONMENTAL VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of `/usr/CODINE` is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *cod_schedd* uses (in the order of precedence):

The name of the cell specified in the environment
variable `COD_CELL`, if it is set.

The name of the default cell, i.e. **default**.

COD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

FILES

`<codine_root>/<cell>/spool/qmaster/schedd`
cod_schedd spool directory

`<codine_root>/<cell>/common/sched_runlog`
cod_schedd trace information

`<codine_root>/<cell>/common/sched_configuration`
cod_schedd configuration

See *sched_conf(5)* for details on the scheduler configuration file.

SEE ALSO

sge_intro(1), *cod_api(3)*, *sched_conf(5)*, *cod_commd(8)*, *cod_qmaster(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

`cod_shadowd` – Sun Grid Engine shadow master daemon

SYNOPSIS

`cod_shadowd`

DESCRIPTION

cod_shadowd is a “light weight” process which can be run on the so called shadow master hosts in a Sun Grid Engine cluster to detect failure of the current Sun Grid Engine master daemon *cod_qmaster(8)* and to start-up a new *cod_qmaster(8)* on the host on which the *cod_shadowd* runs. If multiple shadow daemons are active in a cluster, they run a protocol which ensures that only one of them will start-up a new master daemon.

The hosts suitable for being used as shadow master hosts must have shared root read write access to the directory `<codine_root>/<cell>/common` as well as to the master daemon spool directory (Default `<codine_root>/<cell>/spool/qmaster`). The shadow master hosts need to be contained in the file `<codine_root>/<cell>/common/shadow_masters`.

RESTRICTIONS

cod_shadowd may only be started from root.

ENVIRONMENT VARIABLES

CODINE_ROOT

Specifies the location of the Sun Grid Engine standard configuration files. If not set a default of `/usr/CODINE` is used.

COD_CELL

If set, specifies the default Sun Grid Engine cell. To address a Sun Grid Engine cell *cod_shadowd* uses (in the order of precedence):

The name of the cell specified in the environment
variable **COD_CELL**, if it is set.

The name of the default cell, i.e. **default**.

COMMD_DEBUG_LEVEL

If set, specifies that debug information should be written to stderr. In addition the level of detail in which debug information is generated is defined.

COMMD_PORT

If set, specifies the tcp port on which *cod_commd(8)* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd(8)* to be used for Sun Grid Engine communication of the *cod_qstd* client resides. Per default the local host is used.

FILES

<codine_root>/<cell>/common

Default configuration directory

<codine_root>/<cell>/common/shadow_masters

Shadow master hostname file.

<codine_root>/<cell>/spool/qmaster

Default master daemon spool directory

SEE ALSO

sgc_intro(1), *sgc_conf(5)*, *cod_commd(8)*, *cod_qmaster(8)*, *Sun Grid Engine Installation and Administration Guide*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

NAME

`cod_shepherd` – Sun Grid Engine single job controlling agent

SYNOPSIS

`cod_shepherd`

DESCRIPTION

cod_shepherd provides the parent process functionality for a single Sun Grid Engine job. The parent functionality is necessary on UNIX systems to retrieve resource usage information (see *getrusage(2)*) after a job has finished. In addition, the *cod_shepherd* forwards signals to the job, such as the signals for suspension, enabling, termination and the Sun Grid Engine checkpointing signal (see *sge_ckpt(1)* for details).

The *cod_shepherd* receives information about the job to start from the *cod_execd(8)*. During the execution of the job it actually starts up to 3 child processes. First a prolog script if this feature is enabled by the **prolog** parameter in the cluster configuration (see *sge_conf(5)*). Second the job itself and third an epilog script if requested by the **epilog** parameter in the cluster configuration. The prolog and epilog scripts are to be provided by the Sun Grid Engine administration and are intended for site specific actions to be taken prior and after execution of the actual user job. See *prolog(5)* or *epilog(5)* for detailed information.

After the job has finished and the epilog script is processed, *cod_shepherd* retrieves resource usage statistics about the job, places them in a job specific subdirectory of the spool directory of *cod_execd(8)* for reporting through *cod_execd(8)* and finishes.

RESTRICTIONS

cod_shepherd should not be invoked manually, but only by *cod_execd(8)*.

FILES

`<execd_spool>/job_dir/<job_id>` job specific directory

SEE ALSO

sge_intro(1), *sge_conf(5)*, *cod_execd(8)*.

COPYRIGHT

See *sge_intro(1)* for a full statement of rights and permissions.

NAME

codcommdcntl – Sun Grid Engine communication agent control command

SYNOPSIS

```
codcommdcntl [ -d | -k | -t level ]  
              [ -gid commprocname ] [ -h[elp] ] [ -p port ] [ -S ]  
              [ -unreg commprocname id ]
```

DESCRIPTION

codcommdcntl can be used to control the behavior of *cod_commd(8)* or to retrieve information from a running *cod_commd(8)*.

OPTIONS

-d

Dump internal structures of the running *cod_commd(8)* process to */tmp/commnd/commnd.dump*. The directory */tmp/commnd* must exist and *cod_commd(8)* must have write access to it. The request is ignored otherwise.

This option is mainly intended for debugging purposes. The functionality of the addressed *cod_commd(8)* is not affected.

-k

Kill the addressed *cod_commd(8)*. Pending communications at the time of a kill request will be discarded immediately, yet the shutdown of a *cod_commd(8)* will not leave the processes being connected to the aborting process in an inconsistent state.

-t level

codcommdcntl establishes a connection to *cod_commd(8)* and displays continuous trace output corresponding the trace level specified by **level**. The output consists of a subset of the trace output displayed if *cod_commd(8)* is invoked with the **-ll** option.

Currently the only trace level being supported is 255.

-gid commprocname

Retrieve communication process identification number of **commprocname**. Sun Grid Engine components which enroll to *cod_commd(8)* to be able to communicate with other Sun Grid Engine processes are registered by *cod_commd(8)* with a unique identification consisting of a name and an identification number. The identification name is identical with the name of the Sun Grid Engine component (e.g. **cod_qmaster** for *cod_qmaster(8)*). The identification number can be retrieved by the **-gid** option.

-help

Prints a listing of all options.

-p commdport

Port number to be used in order to address *cod_commd(8)*.

-S

Forces secure ports to be used for communication between *cod_commds* and between other Sun Grid Engine components and the *cod_commds*. This requires all Sun Grid Engine daemons to be started with root permission and the client programs to be configured set-uid root. In turn, it ensures that unauthorized communication is prohibited for non-root accounts.

-unreg commprocname id

Unregister Sun Grid Engine component **commprocname** registered with Id **id** to *cod_commd(8)* (see the **-gid** above for a description of **commprocname** and **id**).

To unregister a Sun Grid Engine component from *cod_commd(8)* can become necessary if a Sun Grid Engine daemon such as *cod_qmaster(8)*, *cod_exeecd(8)* or *cod_schedd(8)* is aborted in an uncontrolled fashion (e.g. by sending the signal SIGKILL via *kill(1)*) and *cod_commd(8)* denies restart of that component with the message **error enrolling to commd: COMMPROC ALREADY REGISTERED**.

The registration facility of *cod_commd(8)* is used to avoid redundant Sun Grid Engine daemons running on the same host. If a Sun Grid Engine component is aborted but unable to unregister from *cod_commd(8)* the registration is kept alive until a time-out of several minutes passes or until the communication process is unregistered manually.

ENVIRONMENTAL VARIABLES

COMMD_PORT

If set, specifies the tcp port on which *codcommdcntl* is expected to listen for communication requests. Most installations will use a services map entry instead to define that port.

COMMD_HOST

If set, specifies the host on which the particular *cod_commd* to be used for Sun Grid Engine communication of *codcommdcntl* resides. Per default the local host is used.

SEE ALSO

sgc_intro(1), *cod_commd(8)*, *cod_execd(8)*, *cod_qmaster(8)*, *cod_schedd(8)*.

COPYRIGHT

See *sgc_intro(1)* for a full statement of rights and permissions.

Index

SYMBOLS

! in .qtask file 207
#!
 determines command interpreter 186
#\$ 187
\$COD_TASK_ID 195
\$HOME 186
\$HOSTNAME 186
\$JOB_ID 186
\$JOB_NAME 186
\$pe_hostfile 189
\$TASK_ID 186
\$USER 186
.cod_request
 private request file 136
.codine_aliases
 file format 134
 user path aliasing 134
.cshrc 53, 174
.kshrc 53
.login 174
.profile 53
.qmon_preferences 217, 234, 238
.qtask 206
 meaning of ! 207
 precedence global/local 207
 syntax 207
.Xdefaults 139, 237
.xinitrc 139, 237
.Xresources 237

/etc/login 174
/etc/services 23, 42, 46
/tmp_mnt
 problems with automounter 134
/usr/CODINE 43
/usr/lib/X11/app-defaults 138
<codine_root> 43
-? qsub option 231
@
 differentiate group from user name 172

NUMERICS

3rd_party 204, 206, 208

A

abort installation procedure 51
-ac
 qsub option 182
-Acal qconf option 112
-acal qconf option 112
access
 file permission 22
access list 37, 124, 171
 add 124
access lists
 show all 124
access lists for PE 146
access restriction

- queues 200
 - type of queue/job 167
- batch job
 - monitoring 55
 - submitting 54
- BSD UNIX 52

C

C

- critical message 158
- c
 - qtsh option 206
- C program integration 206
- C qsub option 187
- c qsub switch 213
- calendar 76
 - add 112
 - delete 112
 - modify 113
 - show 113
- calendar management 109
- calendar_conf 109
- capacity
 - available 88
- capacity planning 88
- cell 37, 42, 44, 188
- checking consistency
 - of a job 182
- checkpoint library 140
- checkpoint process hierarchies 141
- checkpointing 38, 140, 182, 211, 213
 - and restarting 189
 - at shutdown of cod_execd 214
 - file system requirements 215
 - kernel level 141
 - migration 212
 - process hierarchies 211
 - queue type 143
 - type of queue/job 167
 - user level 140, 211
- checkpointing directory 215
- checkpointing environment 38, 141
- ckpt_dir 215
- clean queue 87
- clean-up procedure
 - for QS-jobs 154
- Clear Error 219
- clear qsub option 136
- cluster 38
 - show configuration 71
- cluster configuration 70
- cod_aliases
 - file format 134
 - global path aliasing 134
- COD_CELL 188
- COD_CKPT_DIR 188
- COD_CKPT_ENV 188
- cod_commd 19, 51
- cod_conf 72
- cod_execd 18, 19, 57
 - kill 68, 69
 - looking for via ps 53
- COD_O_HOME 188
- COD_O_HOST 188
- COD_O_LOGNAME 188
- COD_O_MAIL 188
- COD_O_PATH 188
- COD_O_SHEL 188
- COD_O_TZ 188
- COD_O_WORKDIR 188
- cod_pe 196
- cod_qmaster 18, 51, 217
 - kill 69
- cod_qstd
 - configuration file 153
 - queueing system transfer daemon 153
 - spool directory 154
- cod_request
 - global default request 136
- cod_schedd 18, 51, 127
 - configuration file 129
 - kill 69
- cod_shadowd 55
- COD_STDERR_PATH 188
- COD_STDOUT_PATH 188
- COD_TASK_ID 188
- CODINE_ROOT 22, 43, 53, 188
- command line user interface 19

- command-line configuration of manager accounts 119
- commd 19
- COMMD_PORT 53
- common
 - access for shadow master 55
 - access permissions 137
- complex 38
 - administrator defined 169
 - display 169
 - display name list 169
 - global 169
 - host 91, 169
 - name column 170
 - queue 169
 - relop column 170
 - requestable column 170
 - shortcut column 170
- complex attribute
 - load parameter 113
- complex attributes
 - consumable 62
 - Default field 104
 - fixed 62
 - FORCED flag 100
 - inheritance 102
- complex_list 167
- complex_values 167
 - in host_conf 100
- Condor 140
- configuration
 - display 71
 - global 70
 - local 70
 - modify 71
- configure queues 234
- configuring hosts 57
- configuring operator accounts from command-line 121
- consistency checking 182, 218
- consumable 88
 - host related values 62
 - information per queue 234
- consumable resource 235
- Consumable Resources 88
- consumable resources 88, 129

- consumables
 - managing disk space 106
- context 182
- Control Slaves 152
- control slaves
 - PE parameter 147
- cq qconf option 87
- critical message 158
- crontab 237
- csh 206
 - aliases 207
- csh, shell 173
- customization
 - qmon 217, 234, 238
- customizing qmon 138, 237
- cwd
 - problems with 134

D

- d qmod option 70, 237
- daemon
 - execution 18, 19, 57
 - master 18, 56
 - scheduler 18
- daemons
 - restart 70
- date 54
- dc
 - qsub option 182
- dcal qconf option 112
- de qconf option 67
- debug mode 159
 - trace output 160
- debugging with dl 160
- Default
 - field in complex configuration 104
- default 88
- default request 135
 - .cod_request 136
 - file format 136
 - files 135
 - qmon 137
 - qsh 137
- default requests

- order of precedence 136
- delete
 - manager 119
 - operator 121
 - queue 87
 - user 124
- delete administrative host 59
- delete calendar 112
- delete execution host 67
- delete submit host 61
- delete_job
 - QS configuration file entry 154
- dependency 178
- dependency of jobs 231
- dh qconf option 59
- directory
 - root 22
- disable
 - force 234
 - permission 234, 237
 - queue with qmod 236
- disable a queue
 - permission to 121
- disable queue 70
- disable queues 233
- disabled queue 109
- disk space
 - and h_fsize 106
 - management via consumables 106
- disk space requirements 45
- disk space requirements of checkpointing 215
- dispatching jobs
 - with generic queue requests 200
 - with named queue requests 200
- DISPLAY 201
- displaying job priorities 128
- dl 160
- dm qconf option 119
- do qconf option 121
- dq qconf option 87
- ds qconf option 61
- du qconf option 124
- dynamic load balancing 140, 212

E

- E
 - error message 158
- e qmod option 237
- e-mail 158
 - at beginning of job 230
 - at end of job 230
 - monitoring jobs 230
 - when job is aborted 230
 - when job is suspended 230
- email 138, 230
 - format of error mail 159
- embed options 177
- embedding of qsub arguments 187
- enabe
 - force 234
- enable
 - permission 234, 237
 - queue with qmod 236
- enable a queue, permission to 121
- enable queues 233
- ENABLE_FORCED_QDEL 231
- enabled queue 109
- ENVIRONMENT 188
- environment
 - checkpointing 38, 141
 - parallel 39
 - variables 188
- environment variables 188
 - for parallel jobs 198
- epilog 77
- epilogue 77
- ernal level checkpointing 211
- Error 182
- error
 - job state 219
- error message 158
- error reporting 160
- example scripts 54
- exec system call 150
- execd 18, 19
- execution daemon 18, 19, 57
 - kill 68, 69
- execution host 18, 47, 164
 - add 67

- configuration with cron 68
- delete 67
- installation procedure 51
- modify 68
- off-line configuration 68
- show 68
- show list 68
- status 68
- execution host configuration
 - complex_values 100
- execution host spool directory 44
- execution hosts 57
- execution method 77
- explicitly suspended jobs 234

F

- f qdel option 231
- f qmod option 231, 237
- f qstat option 227
- f qstat option with -qsi 157
- fair-share-scheduling 199
- fault tolerance and checkpointing 211
- FIFO 125
- file access permission 46
- file access permissions 22
- file handling
 - administrative user 46
- file size limit
 - h_fsize 107
- first-in-first-out 125, 127, 199
 - unfair scheduling 128
- fixed complex attributes 62, 235
- floating licenses
 - management of 99
- Force 217
- force
 - qmod 237
 - suspend, resume, disable, enable 234
- FORCED
 - flag for complex attributes 100
- format
 - default request file 136
 - messages file 158

G

- getrusage 140
- global complex 169
 - load parameters 113
- global configuration 70
- gmake 208
 - j 210
- group 38

H

- h_fsize
 - disk space management via 106
 - hard files size limit 107
- halt Sun Grid Engine 69
- hard request 194
- hard resource requirements 38
- hold
 - user 178
- hold back job 217
- HOME 188
- home directory path 188
- host 38
 - add administrative 59
 - add execution 67
 - add submit 61
 - administration 18, 57
 - configuration with qmon 58
 - delete administrative 59
 - delete execution 67
 - delete submit 61
 - execution 18, 57, 164
 - execution status 68
 - master 18, 56, 164
 - modify execution 68
 - overview on type of 163
 - show administrative 60
 - show execution 68
 - show execution list 68
 - show submit 62
 - submit 18, 57, 165
- host complex 91, 169
 - load parameters 113
- host file for PEs PE
 - host file 150

- host file for PVM 150
- host object 57
- host_conf 164
 - complex_values entry 100
- HOSTNAME 188
- hostname 167

I

- I
 - info message 158
- id
 - equivalent user 42
- identical user-ids 137
- index
 - of array job 195
- info message 158
- inherit 205, 208, 209
- inheritance of complex attributes 102
- Initial State 76
- inst_codine 51
- install_execd 25
- install_qmaster 24
- installation
 - accounts 22
 - as non-root 24, 25
 - as root 24, 25
 - with root account 46
 - with unprivileged account 46
- installation account 46
- installation directory 43
- installation kit 50
- installation procedure 50
 - abort 51
 - execution host 51
- integration
 - of C programs 206
 - of Java programs 206
- interactive
 - qmake usage 210
 - type of queue/job 167
- interactive job handling 201
- interactive jobs 176
 - default requests 137
 - submitting with qsh 203

- interactive jobs 200

J

- j
 - gmake option 210
 - qmake option 210
 - qstat option 182
- j qacct option 140
- Java program integration 206
- job 38
 - array 186, 194
 - array index 195
 - array task 195
 - context 182
 - error state 219
 - explicitly suspended 234
 - hold 217
 - interactive handling 201
 - monitoring 55
 - monitoring with qstat 227
 - not scheduled 126
 - notify 178
 - parallel 39
 - pending 199
 - pending reasons 126
 - spooling 200
 - submit with qsub 189
 - submitting 54
 - verify 182
- job array 38
- job class 38
- job dependencies 231
- job dependency 178
- Job is first task 147
- job priorities
 - assigning 128
 - displaying 128
- job priority 127
 - value range 127
- job slots 233
- job_finished
 - QS configuration file entry 155
- JOB_ID 188
- job_id 55
 - range of 188

- retrieve with qstat 230
- JOB_NAME 188
- job_status
 - QS configuration file entry 155
- Just verify 182

K

- kej qconf option 68, 69
- kernel level checkpointing 141
- kill
 - execution daemon with jobs 68, 69
 - master daemon 69
- kill scheduler daemon 69
- km qconf option 69
- K-multiplier 193
- k-multiplier 193
- ks qconf option 69
- ksh, shell 173

L

- l qacct option 139
- l qstat option 228
- l qsub option
 - for parallel job 197
- LAST_HOST 188
- limit
 - h_fsize 107
 - per job 107
 - per process 107
- limits
 - per queue slot 235
- list of
 - managers 173
 - operators 173
 - owners 173
- load 130
 - adjustment 125, 130, 131
 - affected by performance 129
 - correction 130
 - reporting 125
 - scaling 125
 - site specific 108

- Load Adjustment 131
- load adjustment 236
- load balancing 170
 - dynamic 140, 212
- load formula 129
- load information 234
- load management 15, 41, 161
- load parameter
 - complex attribute 113
 - virtual_free 104
- load parameters 235
 - adjusted by number of processor 129
 - site specific 129
- load scaling factors 129
- load sensor interface 108
- load_sensor_command
 - QS configuration file entry 155
- load_sensor_file
 - QS configuration file entry 155
- load_thresholds 129
- local configuration 70
- logfile
 - messages 158
- login_shells
 - configuration parameter 174
- logins necessary to use Sun Grid Engine 122, 137
- login-shell 174
- LOGNAME 189

M

- m
 - qsub option 219
- M qsub option 230
- m qsub option 230
- MAIL 188
- mail 70
- make 208
- manager 38, 117
 - add 119
 - delete 119
 - display list 173
 - show 119
- manager accounts

- commd-line configuration 119
 - configuring with qmon 118
- manager, user category 162
- managing disk space 106
- manipulate queues 75
- mapping
 - of Sun Grid Engine/QS job-ids 154
- master
 - as administration host 52
 - as execution host 51
 - as submit hosts 52
- master daemon 18, 56
 - kill 69
- master host 18, 47, 56, 164
 - restriction 51
- master installation procedure 50
- master queue 182
- master spool directory
 - access for shadow masters 55
- maxjobs 126, 130
- maxujobs 126, 130
- Mcal qconf option 113
- mcal qconf option 113
- mconf qconf option 71
- Me qconf option 68
- me qconf option 68
- memory 193
 - multipliers 193
 - requirements for checkpointing 215
- memory oversubscription
 - avoid 103
- message passing 197
- Message Passing Interface 145
- messages
 - logfile 158
- messages file
 - format 158
- migr_load_thresholds 129, 212
- migrate jobs 140
- migration 38
 - reasons 212
- migration of checkpointing jobs 212
- min_cpu_interval 214
- M-multiplier 193
- m-multiplier 193

- modify calendar 113
- modify execution host 68
- modify pending jobs
 - job
 - modify 217
- modify queue 87
- monitor queues 75
- monitoring a QS 157
- monitoring by electronic eail 230
- monitoring jobs with qstat -f 227
- Motif GUI 138
- MPI 145, 147, 152, 196
- MPICH 152
- mpirun 147
- Mq qconf option 87
- mq qconf option 87
- multi CPU machines 129
- multi processor systems 129
- multipliers 193

N

- N
 - notice message 158
- name
 - in complex definition 170
- navigating through the Sun Grid Engine system 163
- network services 46
- newgrp 172
- NFS Network File System 215
 - problems with 134
- NHOSTS 189
- nice 128
- NIS 23, 42, 46, 53
- notice message 158
- Notify 76
- notify a job 178
- now 205
 - qlogin option 201
 - qrsh option 201
 - qsh option 201
 - qsub option 201
- NQS 156, 189
- NQUEUES 189

- NSLOTS 189
- number of jobs
 - restricted 130
- number of processors
 - adjusts load 129

O

- ob sorting 125
- operator 39, 117
 - add 121
 - delete 121
 - display list 173
 - show 121
- operator accounts
 - command-line configuration 121
 - configuring with qmon 120
- operator, user category 162
- option embedding 177
- OSF/1 Motif 138
- oversubscription 88
- overview on host types 163
- overview on the Sun Grid Engine system 163
- owner 39
 - display list 173
- owner, user category 162
- owner_list 167
- owners of queues 117

P

- P column in qstat output 199
- p qalter option 128
- PAM-CRASH 99
- parallel
 - type of queue/job 167
- parallel computing 145
- parallel environment 39, 145
 - access lists 146
 - access restrictions 197
 - allocation rule 147
 - displayed with qmon 196
 - host file 150
 - stop procedure 147

- submitting jobs to 196
- Parallel Environment Configuration 196
- parallel environment interface 171
- parallel job 39, 189
 - environment variables 198
 - resource requirement 197
- parallel jobs 196
- parallel virtual machine 145
- PATH 188, 189
- path
 - default shell search 189
- path aliasing
 - file format 134
 - file interpretation 135
- PE 189
 - access lists 146
 - access restrictions 197
 - accounting 152
 - allocation rule 147
 - control slaves parameter 147
 - displayed with qmon 196
 - parallel environment 145
 - process control 152
 - resource limits 152
 - start-up procedure 147, 150
 - stop procedure 147, 151
 - submitting jobs to 196
 - tight integration 152
- pe 210
- pe qsub option 197
- PE start-up procedure 150
- PE_HOSTFILE 189
- pending
 - reasons 126
- PENDING JOBS 228
- pending jobs 199, 228
 - consistency checking 218
 - priority value 199
- per job limits 107
- per process limits 107
- preferences
 - qmon 217
- performance
 - affecting load 129
- permission
 - file access 22

- to suspend, resume, disable, enable 234
 - to suspend, unsuspend, disable, enable 237
- permission
 - file access 46
- physical memory
 - and virtual_free 103
- please
 - display properties 167
- policy 39
 - queue selection 127
 - scheduling 127
- preferences
 - qmon 234, 238
- prefix string 177
- price performance ratio 129
- primary
 - master host 55
- priorities
 - assigning 128
 - displaying 128
- priority 39
 - job 127
 - value of pending jobs 199
 - value range 127
- problems startung shadow qmaster 56
- process control
 - and PEs 152
- process hierarchy
 - checkpointing 141, 211
- processor number
 - adjusts load 129
- processors 167
- prolog 77
- prologue 77
- properties
 - queue 170
- properties of a queue 167
- properties of queues 170
- ps
 - to look for cod_execed 53
 - to look for qmaster 52
- PVM 145, 147, 152, 196
- pvm 145
- PVM host file 150

Q

- qacct 19
 - generating accounting statistic 139
 - j 140
 - l 139
 - referenceing resoure requirements 139
- qalter 19, 217
 - assigning job priorities 128
 - consistency checking 218
 - context 182
 - l option 100
 - p 128
 - scheduler monitoring with 126
 - w 126
- qconf 19
 - Acal 112
 - acal 112
 - ae 67
 - ah 59
 - ao 121
 - Aq 87
 - aq 87
 - as 61
 - au 124, 144, 145, 149, 150
 - cq 87
 - dcal 112
 - de 67, 68
 - dh 59
 - displaying complex 169
 - displaying complex name list 169
 - displaying operator accounts 173
 - displaying trusted hosts 164
 - do 121
 - dq 87
 - ds 61
 - du 124
 - kej 68, 69
 - km 69
 - ks 69
 - maintain calendar configuration 112
 - maintain manager list with 119
 - maintain operator list with 121
 - mcal 113
 - mconf option 71
 - Me 68
 - modify configuration 71
 - Mq 87
 - mq 87

- sc 169
- scal 113
- scall 113
- scl 169
- sconf 71
- se 68
- se option 164
- sel 68, 164
- setting up administration hosts 57
- setting up submit hosts 57
- setting up trusted hosts 57
- sh 60, 164
- show configuration 71
- sm 119
- so 121
- sp 196
- spl 196
- sq 87
- sql 88
- ss 62, 165
- su 124, 172
- sul 124, 172
- tsm 127
- qconf -ah 52
- qconf -am 119
- qconf -as 52
- qconf -dm 119
- qdel 19, 195
 - f 231
- qhold 20, 195
- qhost 20, 68, 164
- qlogin 20, 200, 204
 - context 182
 - now 201
- qlogin vs. qrsh 204
- qmake 20, 208
 - batch usage 210
 - inherit 209
 - interactive usage 210
 - j 210
 - pe 210
 - syntax 209
- qmake option 209, 210
- qmaster 18, 56
 - looking for via ps 52
- qmaster spool directory 44
- qmod 20, 70, 195

- d 70, 237
- disable queue 70, 236
- e 237
- enable queue 236
- f 231, 237
- force 237
- s 231
 - s qmod option 237
- suspend queue 236
- suspending a queue 237
- unsuspend queue 236
- us 231, 237
 - with crontab or at 237
- Qmon 138, 139
- qmon 18, 20, 138
 - and embedded script arguments 187
 - configuring manager accounts 118
 - configuring operator accounts 120
 - customization 217, 234, 238
 - customizing 138, 237
 - default requests 137
 - displaying parallel environments 196
 - host configuration 58
 - preferences 217, 234, 238
 - update 234
- Qmon resource file 237
- qname 167
- qresub 20
- qrls 20, 195
- qrsh 20, 200, 204
 - inherit 205
 - now 201, 205
 - syntax 205
 - verbose 205
 - within qtcsh 206
- qrsh option 205
- qrsh vs. qlogin 204
- qrshmode 208
- QS 153, 198
 - monitoring 157
- QS command procedures
 - rules 155
- QS interface configuration file 153
- qs_args qsub option 198
- qselect 20
 - l option 100
- qsh 20, 201

- context 182
- default requests 137
- l option 100
- now 201
- submitting interactive jobs 203
- QSI 153, 198
 - configuration file 154
- QSI command procedure example 156
- QSI configuration file example 156
- qsi qstat option 155, 157
- qstat 18, 20, 195
 - displaying job priorities 128
 - f monitoring jobs with 227
 - f option with -qsi 157
 - j 182
 - l 228
 - l option 100
 - monitor batch jobs 55
 - monitoring jobs with 227
 - P column 199
 - PENDING JOBS 228
 - qsi option 155, 157
 - qtype column 228
 - r 228
 - resource requirements 228
 - retrieve job_id 230
 - state column 227
 - states column 228
 - used/free column 228
- qsub 18, 21
 - ? 231
 - ac 182
 - arguments in scripts 187
 - C 187
 - c 213
 - clear 136
 - context 182
 - cwd for checkpointing jobs 215
 - dc 182
 - l for parallel job 197
 - l option 100
 - M 230
 - m 230
 - m a 219
 - now 201
 - options, read from file 194
 - overriding embedded options from command line 194
 - pe 197
 - qs_args 198
 - r option 76
 - requesting attributes 170
 - submit batch job 54
 - submit jobs with 189
 - submitting a parallel job 196
 - submitting generic requests 171
 - submitting to named queue 189, 190
 - t 195
 - V for parallel job 198
 - v for parallel job 198
- qtask file 206
- qtcs 21, 206
 - aliases 207
 - c 206
 - shell builtin command qrshmode 208
 - usage 206
- qtype
 - of queue/job 167
 - qstat column 228
- quantity syntax 193
- QUEUE 189
- queue 19, 39
 - add 87
 - attributes 235
 - clean 87
 - complex list 167
 - complex values 167
 - configuration 234
 - configuration template 87
 - delete 87
 - disable 233
 - disable with qmod 236
 - disabled by calendar 109
 - display list 166
 - enable 121, 233
 - enable a 121
 - enable with qmod 236
 - enabled by calendar 109
 - manipulate 75
 - master 182
 - modify 87
 - monitor 75
 - owner 117, 121, 162
 - owner_list 167
 - processors 167
 - properties 170

- resumed by calendar 109
- selection by seq_no 130
- selection policy 127
- shell parameter 186
- show 87
- show list of 88
- slave 182
- slot limits 235
- slots 167
- suspend 121, 233
- suspend with qmod 236
- suspended by calendar 109
- type for checkpointing 143
- unsuspend 121
- unsuspend with qmod 236
- unsuspended by calendar 109
- user access list 167
- queue calendar 109
- queue complex 169
 - load parameters 113
- queue sorting 125
- queue_conf 107
- queue_sort_method 130, 200
- queueing_system
 - queue configuration entry 154
- queuing system interface 153, 198
- queuing_system
 - QS configuration file entry 154
- queuing_system_up
 - QS configuration file entry 154

R

- r
 - qstat option 228
- r qsub option 76
- range of job_id 188
- REAL 139
- reasons for not scheduling jobs 126
- redirection
 - stderr 55
 - stdout 55
- relation operation 170
- release job
 - job
 - release 217

- relop in complex definition 170
- remsh 204
- REQUEST 189
- request
 - hard 194
 - name 189
 - soft 194
- requestable
 - in complex definition 170
- requirements
 - hard 38
 - soft 39
- rerun
 - default queue policy 76
 - jobs 76
- resource 39
 - allocation algorithm 194
 - usage 140
- resource capacity 235
- resource consumption information 234
- resource customization template 138
- resource limits
 - and PEs 152
- resource requirement
 - for parallel job 197
- resource requirements
 - hard 38
 - referencing with qacct 139
 - soft 39
- resource requirements with qstat 228
- resources
 - available on host 62
 - x-windows 138
- restart files 140
- restart mechanism 211
- restart Sun Grid Engine daemons 70
- RESTARTED 189
- restarted checkpointed jobs 189
- restarted job script 213
- restrict number of jobs 130
- resume
 - force 234
 - permission 234, 237
- resume job method 77
- resume queue 233

- resumed queue 109
- rlogin 200, 204
- root
 - installation as 24, 25
- root account 22, 46
- root directory 22, 43
- rsh 54, 200, 204
- rules
 - for QS command procedures 155

S

- s qmod option 231
- sacle load 129
- Save 217
- sc
 - qsub option 182
- sc qconf option 169
- scal qconf option 113
- scall qconf option 113
- schedd 18
- schedd spool directory 44
- schedd_conf 129, 130
- schedd_job_info 218
- Scheduler Configuration 131
- scheduler configuration file 129
- scheduler daemon 18
 - kill 69
- scheduler monitoring 127
 - qalter 126
- scheduling
 - activities 124
- scheduling policy 127
- scheduling procedure 126
- scl qconf option 169
- sconf 71
- sconf qconf option 71
- script embedding 187
- se qconf option 68, 164
- sel qconf option 68, 164
- selecting queues by seq_no 130
- seq_no 130, 200
- seqno 130

- sequence number 125
- services 42, 53
- services database 23, 46
- setrlimit 107
- settings.csh 53
- settings.sh 53
- sh qconf option 60, 164
- sh, shell 173
- shadow master
 - access to common directory 55
 - hostname file 55
- shadow master host 47, 55
- shadow qmaster
 - problems starting 56
- shadow_masters 55
- shadow_masters file 47
- SHELL 188, 189
- shell
 - queue parameter 186
 - scripts 173
- Shell Start Mode 76
- shell_start_mode 185
- short-cut
 - for attribute name 100
- shortcut
 - in complex definition 170
- show
 - all access lists 124
 - managers 119
 - operators 121
 - queue configuration 87
 - queue list 88
 - user access list 124
- show administrative hosts 60
- show calendar 113
- show configuration 71
- show execution host 68
- show execution host list 68
- show submit hosts 62
- shut-down Sun Grid Engine 70
- shutdown the PE 151
- site specific load parameters 129
- site specific load information 108
- Skip 182
- slave queue 182

- Slot-Limits/Fixed Attributes 235
- slots 167
- sm qconf option 119
- so qconf option 121
- soft request 194
- soft resource requirements 39
- sp qconf option 196
- space sharing 103, 104
- spl qconf option 196
- spool directories 44
- spool directory
 - of cod_qstd 154
- spooling jobs at qmaster 200
- sq qconf option 87
- sql qconf option 88
- ss qconf option 62, 165
- standard error 174
- standard output 174
- start job method 77
- start-up procedure 147
- state
 - column in qstat output 227
- states
 - qstat column 228
- STDERR
 - of QS submit command 156
 - of QS-jobs 154
- stderr redirection 55
 - redirection
 - stderr 159
- STDOUT
 - of QS submit command 156
 - of QS-jobs 154
- stdout redirection 55
- stop procedure 147
- stop procedure for PE 151
- stty 53, 54
- su qconf option 124, 172
- submit
 - QS configuration file entry 154
 - with qsub 189
- submit host 18, 48, 165
 - add 52, 61
 - delete 61
 - show 62

- submit hosts 57
 - setting up 57
- sul qconf option 124, 172
- Sun Grid Engine root directory 43
- Sun Grid Engine startup procedure 70
- supercomputer 153, 198
- suspend
 - a queue 237
 - a queue, permission to 121
 - force 234
 - permission 234, 237
 - queue with qmod 236
- suspend job explicitly 234
- suspend job method 77
- suspend queues 233
- Suspend Thresholds 79
- suspend thresholds 79
- suspended queue 109
- suspension 39
- swap space 103
 - and virtual_free 103
- swapping 103
 - avoid 103
- syntax
 - time value 193
- SYSTEM 139
- SYSV UNIX 52

T

- t
 - qsub option 195
- task 195
- TCP 23
- tcsh 173, 206
- telnet 200, 204
- template
 - for queue configuration 87
 - for resource customization 138
- temporary directories 137
- terminal connection of scripts 174
- terminal control for batch jobs 204
- terminal I/O for batch jobs 204
- terminate job method 77

- tight PE integration 152
- time value syntax 193
- time zone 189
- TMP 189
- TMPDIR 189
- trace output
 - debug mode 160
- TRANSFER 153, 198
- transfer 167
- transfer queue 153, 198
- transfer_down
 - qstat queue status output 154
- transfer_queue
 - QS configuration file entry 154
- trusted hosts 119
 - setting up 57
- tsm qconf option 127
- tty
 - s option 54
- type
 - of queue for checkpointing 143
 - queue configuration entry 198
- TZ 189
 - time zone 188

U

- unix_behavior 186
- unprivileged account 46
- unsuspend
 - force 234
 - permission 234, 237
 - queue with qmod 236
- unsuspend a queue
 - permission to 121
- unsuspend queues 233
- unsuspended queue 109
- update qmon 234
- us qmod option 231, 237
- usage information 140
- usage within qmake 208
- used/free
 - qstat column 228
- USER 139, 189

- user 39, 117
 - categories 117, 162
- user access list 167
 - add 124
 - delete 124
 - show 124
- user access lists
 - show all 124
- user access lists for PE 146
- user access permissions 171
- user defined complex
 - load parameters 113
- user hold 178
- user id
 - equivalent 42
- user interface
 - command line 19
- user level checkpointing 140, 211
- user sort 125
- user_lists 167, 172
- user_sort 128, 199
- user-ids
 - identical 137
- userset 39
- utilization 139

V

- V qsub option for parallel job 198
- v qsub option for parallel job 198
- variables
 - environment 188
- verbose
 - qrsh option 205
- Verify 182
- Verify flag 218
- verify job 182
- verify job consistency 218
- vi editor 67, 68, 87, 144, 149
- virtual_free 103
 - load parameter 104

W

W

- warning messages 158
- w option to qalter 126
- Warning 182
- warning messages 158
- Why 218
- working directory
 - temporary 137
- workload information 171

X

- XAPPLRESDIR 237
- xrdb 139, 237
- xterm 70, 201
 - for interactive jobs 203
- xuser_lists 167, 172
- x-windows
 - resources 138