Multithreaded NFS Delivers Performance Gains for Solaris 2

CASE STUDY

NFS®, Sun's distributed file system, is an essential component of the advanced client/server functionality offered by SunSoft's Solaris™ software environment. In 1990, as SunSoft moved toward the release of Solaris 2, and support for multiprocessor systems; it became necessary to create a "thread-safe" version of NFS. (Because the kernel itself is multithreaded, all the code in it must be thread-safe — meaning that multiple copies of it can run simultaneously.) Multithreading turned out to be key in improving the performance of both the NFS server and client software, providing a degree of concurrency which the original single-threaded scheme could not.

Performance Bottlenecks

The use of a single, unthreaded NFS daemon had resulted in major I/O bottle necks in the earliest version of NFS, especially in the area of disk access. An NFS daemon is a server-side process that provides file service. It operates by waiting for a request to come in, performing some CPU work, scheduling some I/O, sleeping while that completes, doing a bit more CPU work, and then going back to sleep, waiting for the next request. A request, in turn, comes in to a server port where it is sent to the transport queue. It awakens the next available daemon, and restarts the cycle. The only way to get I/O concurrency in SunOS™ 4.x was to run multiple NFS daemons. This required significant overhead caused by context switching between the processes, along with high costs for the cross-process locking primitives.

Finding the Right Approach

In the first attempt to multithread the NFS server, the existing SunOS 4.x software model was used, and locks were added everywhere that was required to make it thread-safe. This approach was unsuccessful, because the algorithms, developed for uniprocessor systems, didn't scale well in a multiprocessor environment. Expensive, and with high overhead, this lock-everything methodology fell considerably short of the desired server performance.

"Next, we went to a completely threads-based model," recalls SunSoft engineer Peter Staubach. "We took off the user-level processes and added code in the kernel to create threads as needed, but still used basically the same locks. This yielded some performance improvement, but not enough to compete in the market place. The problem was two-fold: the algorithms still could not scale to accommodate more processors and memory, and there was still too much global data that required locking. If you're handling 3,000 requests per second, you can't use the same kind of linear queue which might very easily process 1000 requests per second."

SunSoft
A Sun Microsystems, Inc. Business

Successfully Multithreading the NFS Server

NFS daemons spend all of their useful lives in the kernel. After a daemon is started, it makes a system call to start processing NFS requests, and it remains in the kernel. Staubach explains: "Concurrent execution requires process contexts, which are easy to create at user-level, but quite messy to do in the kernel. Because all threads have to share the same end point, basic UNIX® semantics work well."

When a request comes in to a server port, an NFS daemon opens the transport end point and delivers a request to a queue in the kernel. The trick is in the use of real user process contexts for the kernel threads. "It is easy to create and destroy kernel threads dynamically, and to scale the number of threads up or down to match the size of the server, with no need to pre-allocate them," says Staubach. "The server just runs, creating potentially hundreds of threads."

"Our goal was to make NFS 'Multithread Hot'" says Staubach, "Which means limiting areas that can't have concurrency to the smallest size possible. We spent a lot of time identifying these areas and tuning the system by modifying algorithms to minimize their use. One thing we did was to use hashing (sending requests to different queues) to make the queues smaller and get each request off its queue more quickly."

It was this, rearchitecting the server to be fully threaded, along with some algorithm tweeks, and minimizing the amount of time spent in critical sections, that gave the desired results. Currently, Solaris 2.4 is well-tuned for multiprocessor scalability, so that performance gains from multithreading are near-linear with the number of CPUs available. The scalability now is limited by overall system overhead, such as file system and transport mechanisms.

Multithreading the NFS Client

NFS client-side code is completely separate from the server-side. SunOS 4.x used a single-threaded client that was slow by today's standards, because it could do only one write at a time. SunOS couldn't do write-behinds (queuing data and transmitting it in an asynchronous process from the writing thread). Instead, BIO daemons would pick up and transmit write-behind requests to the server. These requests used Remote Procedures Calls (RPCs), whose wait times are longer than disk I/O wait times. While awaiting the RPC response, another BIO daemon could transmit another write request to the same server, relying on the server's multiple NFS daemons to be processing several requests at the same time.

For Solaris 2, the client also had to be made thread-safe. "As with the server, the client team started out trying locking techniques with the original code," says Staubach. "But the NFS client is more complex than the server. A file system client is a stateful object, and has to know much more information, such as which files are open, etc. It also handles caching. Again, we were trying to adapt uniprocessor algorithms which had no lock requirements originally. They were hard to convert, and the locking methods proved to be clumsy and too complex."

Here too, the solution was to start over with new MT algorithms. As with the server-side, the team was able to create and destroy threads inside the kernel. That meant they could dispense with the old BIO daemon model and use threads in the kernel, created as needed on a per-file-system basis rather than for the whole client. Now, rather than being limited by the number of BIO daemons, the client could use as many threads as required at any given time.

Performance and Scalability

"Actually, much of the kernel code was fine, and only the most critical fifteen percent of it had to be rewritten for threads — principally the sections dealing with the queue," notes Staubach. In the threaded version of NFS, the multitude of threads is transparent to the user. Optimum scalability at present is about 10 CPUs.

Staubach concludes, "While we don't have a standard benchmark for client performance, running the threaded NFS in a variety of application contexts has resulted in significantly better performance than with the older version. We know we are making better use of multiprocessor environments, and we have much better maintainability with clean, simple algorithms and asynchronous threads that have simpler functions. Multithreading allowed us to accomplish a number of goals, not the least of which was to enable NFS users to scale their configurations to their workload."

Sun Internal NFS Performance Tests

	SS10/41 (1 CPU)	*SC2000 (8CPUs)*
SunOS 4.1 (single threaded)	100	N/A
Solaris 2.2 (multithreaded)	102	248
Solaris 2.3 (multithreaded)	119	752
Solaris 2.4 (multithreaded)	~150	~950

For more information about how you can take advantage of multithreading, please send email to Threads-Hotline@Sun.COM, or call 415 336-3032.

SunSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue, Mountain View, CA 94043-1100 USA • (415) 960-3200 • Fax (415) 961-6070
Printed in USA 11/94 94336-001