

THE VIEW

How Lotus Notes and Domino Professionals Succeed™

2008 Exclusive Reprint

Tuning Domino for a high-throughput processor

Eric Sosman, Staff Engineer, Sun Microsystems, Inc.

Tuning Domino for a high-throughput processor

by Eric Sosman



Eric Sosman
Staff Engineer
Sun Microsystems, Inc.

Eric Sosman has four decades of experience as a software engineer. For the last ten years, he has served as technical liaison between Sun and the Lotus engineering teams, and as Sun's delegate to the NotesBench Consortium. His primary focus is Domino's performance on Solaris. He was the very first to test Domino on the UltraSPARC T1 "Niagara" processor and he contributed snippets of code to Domino. Eric has spoken at various conferences, writes the annual "Domino on Solaris: Common Tuning Tips" guide, and is a member of the Computer Society of the IEEE.

The widening gap between central processing unit (CPU) speed and main memory speed has challenged high-performance computing for more than a quarter century. CPU designers have used more and more complex strategies to reduce the speed penalty of memory transactions, and they have employed more and more levels of larger and more sophisticated cache memories. Yet in all but the most "regular" computations, fast CPUs still seem to idle for too many cycles while they wait for memory to catch up.

In 2005, Sun Microsystems introduced the UltraSPARC® T1 (code-named "Niagara") processor, which takes a completely different approach to dealing with memory latencies. Rather than trying to avoid memory stalls, a strategy that has not been wildly successful, the T1 accepts the fact that memory delays are inevitable and tries to improve throughput by doing useful work instead of simply waiting. The strategy has been a great success for applications such as IBM® Lotus® Domino®, which is both highly parallel and memory-intensive. Domino's out-of-the-box performance is excellent, but there are still opportunities to tune both Domino (6.5 and up) and Sun™ Solaris™ for even better throughput. In this article, I briefly describe the UltraSPARC T1 and its 2007 successor, the UltraSPARC T2, and offer some approaches for tuning Domino and Solaris to take advantage of the thread-rich environments that these processors provide.

The Niagara processors

The computer industry rewards memory manufacturers for high density and low cost, while it rewards CPU manufacturers for speed. It is certainly possible to build very fast memory parts, but we sacrifice both density and cost when we do so. High-speed cache memories take a lot of chip space, and there's a reason for the dollar sign in notations like "D\$" for a data cache. This is the economic origin of the speed gap between memory and

CPU, and unless the industry as a whole makes an unlikely and drastic shift in its goals, the gap will remain and will probably grow wider.

The designers of the UltraSPARC T1 felt that trying to paper over the speed gap was a losing game, where the next increase in CPU clock speed or in memory density would erase even a momentary gain. Moore’s Law describes an increase in per-chip transistor count that raises the question: What is the best way to employ all those transistors? The T1’s designers answered this question with a chip architecture that allows the processor to continue doing useful work even while waiting for memory responses.

The T1 has eight processor cores, each supporting four hardware execution contexts (called “strands,” as the hardware analogue to software threads). Each core switches control from strand to strand at every clock cycle: Strand S0 gets one cycle, S1 gets the next, then

S2, S3, and back to S0 again. If strand S1 stalls while waiting for memory, it simply drops out of the rotation, which then looks like S0, S2, S3, S0. The other three strands continue working while S1’s memory transaction completes. When that happens, S1 rejoins the rotation and resumes its work. The whole core becomes idle only when all four strands stall at the same time, or when Solaris parks them because there’s no work to be done. This way, the time a strand spends in a memory stall is not wasted, because the other strands are making use of the core. See **Figure 1** for a visual representation of this principle.

Like any engineering strategy, the T1 approach has some trade-offs. The most evident drawback is that a core with four strands running can give only one-quarter of its cycles to each strand; this means that a 1,200 MHz processor core runs each of its four strands at about 300 MHz. (Because memory stalls are

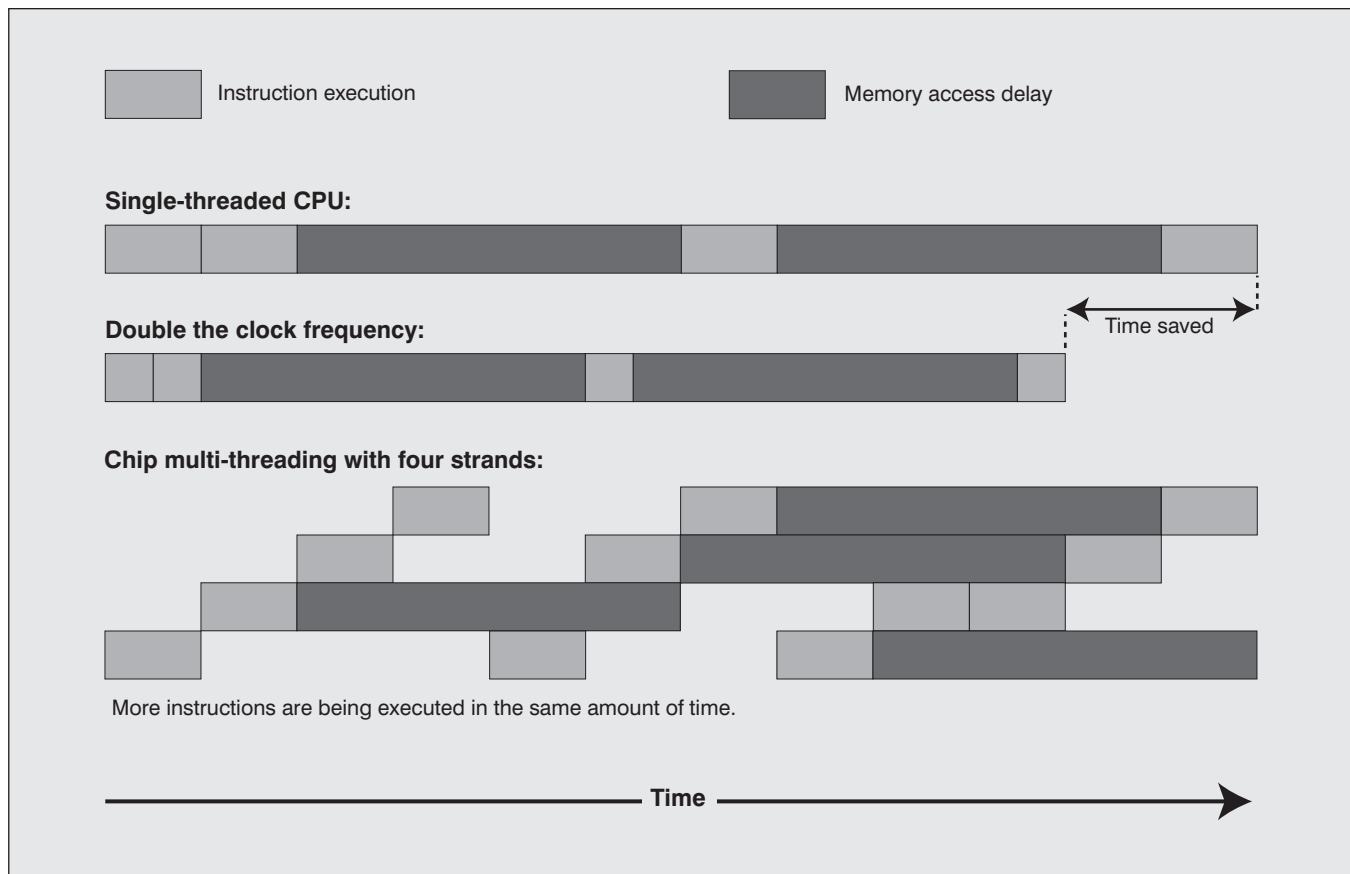


Figure 1 Comparison of standard CPU usage with multi-threading architecture

so much less disruptive, it's not entirely accurate to estimate the CPU's overall speed this way, but it's in the right ballpark.) Even when three strands are stalled or parked and the remaining strand gets all the core's attention, the core's single-threaded performance is only mediocre in comparison to intricate traditional designs. Therefore, the best way to maximize throughput from a Niagara system is to keep all the strands as busy as possible: Run many parallel activities and prevent a single-threaded activity from becoming a bottleneck. Through the rest of this article, I provide some hints for promoting parallelism and avoiding bottlenecks in Domino.

The UltraSPARC T2 "Niagara 2" processor, introduced in 2007, pursues the same strategy somewhat further. Like the T1, it is an eight-core CPU, but it supports eight strands per core instead of four. It also provides two instruction-execution pipelines per core to increase the core's internal parallelism. Its on-chip memory cache is one-third larger than the T1's, and the paths for network and other input/output (I/O) are more tightly integrated onto the chip to reduce latencies. Some improvements, such as a vastly better floating-point system and integrated cryptography, are not of much importance to Domino, although they're of great interest to some other applications. All in all, Domino running on a T2 processor experiences about 1.8 – 1.9 times the throughput it would get from a T1 at the same clock speed. The same tuning recommendations apply for both models of the Niagara family.

Tuning for parallelism

The architectures of the UltraSPARC T1 and T2 processors favor highly parallel applications, and Domino has all the right characteristics to benefit. A single Domino partition runs multiple processes that correspond (roughly) to its server tasks, and most of those processes run multiple threads of execution. Add all the processes together and you find that even a relatively barebones Domino partition runs close to two hundred threads. Turn on some Web services and add a few Internet Message Access Protocol (IMAP) clients and the like, and the thread count climbs toward three

hundred or more. Run multiple Domino partitions on a single machine — some sites run in excess of two dozen partitions — and the total thread count can approach six or seven thousand.

Most of the time, most Domino process threads are idle, waiting for something to do. There's a thread that monitors the network interface, waiting for the next incoming message. There are router threads waiting to dispatch newly arrived mail. There are threads that manage the transaction logs, threads that run agents at scheduled times, even threads to watch for the shutdown of the Domino server! Even so, if just one-tenth of Domino's threads are active at any moment, each partition has twenty to thirty threads that need attention from the CPU. With all these threads, there is plenty of built-in parallelism in Domino that the T1 or T2 can exploit for you; therefore, the focus of tuning is to eliminate bottlenecks.

Update Solaris

The first and most important task might also be the most banal: Make sure you're using an up-to-date version of Solaris. Niagara's massive parallelism changes the equation not only for applications, but also for the task scheduler that Solaris uses to decide which software threads run on which hardware strands and when. The Solaris 10 edition that was current in 2005, when the UltraSPARC T1 was introduced, simply treats the thirty-two hardware strands as 32 independent execution units. This edition is not equipped to understand that each strand is not truly independent, because each quartet of strands shares the resources of a single core. Updating the scheduler to account for the interactions among cores and strands was one of the first improvements made to Solaris after the T1 was released, and further improvements were made as Solaris' developers gained experience with Niagara. All the improvements are available as patches and are incorporated in all recent Solaris versions, so you can easily update by installing from recent media or by visiting <http://sunsolve.sun.com/> and installing the current recommended patch cluster (which is a good idea in any case). Don't limit your parallelism by running a mint-condition vintage 2005 Solaris!

The scheduler, reloaded

Solaris' task scheduler tries to assign software threads to hardware strands in a way that maximizes overall system throughput. Because no one algorithm is appropriate for all situations, Solaris supports several scheduling classes that use different strategies to assign CPU resources. I recommend that you run Domino in a scheduling class other than the default; in this section, I explain the reasoning behind that recommendation and how to make the change.

I've mentioned that Domino is highly parallel, but parallelism has a price. An old joke says that if one man can dig a posthole in three minutes, then 180 men can dig the same hole in one second flat. The fallacy is in the coordination: The 180 workers simply could not crowd around the same hole, synchronize their shovels to keep out of each others' way, dispose of their shoveled dirt without whacking each other in the eye, and so on. So, too, with Domino's highly parallel threads: The activities must be carefully synchronized if the whole effort is not to degenerate into unproductive flailing.

By default, Solaris assigns most threads to the time-sharing (TS) or interactive (IA) scheduling classes, which use heuristics (approximation algorithms) to track each thread's execution profile and estimate its position on a scale of CPU usage. This usage continuum runs from compute-bound threads that soak up pretty much all the CPU time they are given to event-driven threads that use the CPU in short bursts and then become idle, waiting for an event like the completion of an I/O operation. Because the scheduler tracks the execution history of each thread, it can update its estimate of the thread's position on the continuum as its behavior changes over time. The basic goals are to give CPU priority to event-driven threads, with the expectation that these threads will finish quickly and resume waiting, and to let the compute-bound threads run while the others wait for their events.

Unfortunately, this egalitarian scheduling strategy meshes poorly with Domino's own coordination of its multiple parallel activities. For example, a Domino thread sometimes voluntarily relinquishes its opportunity to use CPU resources in favor of another Domino thread; however, if Solaris judges the first

thread to be event-driven and the second to be compute-bound, Solaris gives the first thread higher priority. That means that when the first thread gives up the CPU, Solaris might give it right back again without giving the second thread a chance to run. Nothing breaks, exactly, but the interaction between Solaris' scheduler and Domino's own scheduling is sub-optimal.

I suggest that you run Domino in Solaris' fixed-priority (FX) scheduling class, which does not track threads' execution profiles or adjust priorities according to the profiles. Instead, it leaves each thread's priority unchanged and assigns equal-priority threads to the CPU in round-robin fashion. This policy blends better with Domino's needs than the default does.

The most straightforward way to run Domino in the FX class is to use the `priocntl` command when starting the server. Use the command as follows, with your normal Domino launch command in place of the italicized phrase:

```
priocntl -e -c FX -m 45 -p 45 server_launch_command
```

This command starts Domino in the FX class with a priority of 45, which is moderately high in the allowable range of 0 through 60.

Note!

You should *not* run Domino at the maximum priority of 60, because if a thread gets into an infinite loop, it monopolizes the CPU and you would have a hard time killing it with the `nsd` command! Leave yourself some priority headroom.

If you use a shell script to start Domino, you can use a slightly different command in the script itself at any time before Domino starts, as follows:

```
priocntl -s -c FX -m 45 -p 45 $$
```

This command puts the shell itself and all of the child processes it runs thereafter, including Domino, into the FX class at priority 45.

If you use the Unix user account that runs Domino for only that purpose, still another approach is to put the second of the above `priocntl` commands in the account's shell startup script, e.g., `~/bashrc` or `~/cshrc`.

For a Unix user account to use any of these commands, you must first authorize it to use `priocntl` this way; Solaris does not let just any user alter scheduling parameters with the freedom that `priocntl` allows. As the root user, execute this command to grant the necessary privilege to the user account that runs Domino:

```
usermod -K defaultpriv=basic,priv_proc_priocntl
username
```

This command grants the specified user all the “basic” privileges, as well as the right to use `priocntl` to modify scheduling parameters, for his or her own applications. Note, though, that this latter privilege is potentially dangerous, because it allows the user to run tasks at a higher priority than the special tasks that are vital to system health. Be sure to grant it only to users who can be trusted to employ it safely.

For more information on scheduling and priority management, use the ‘`man -s1 priocntl`’ command. Background information on Solaris 10 privileges can be found with ‘`man -s5 privileges`’, while ‘`man -s1m usermod`’ and ‘`man -s1m smc`’ discuss tools to administer them.

Use your memory

You can enhance parallelism by using memory strategically, because when caches, buffer pools, and the like perform well, threads wait for them less and get out of each other's way more quickly. This tuning suggestion has two parts: One concerns memory devoted to Domino, and the other concerns memory devoted to the file system.

Tuning Domino's memory usage

Domino uses many internal caches, buffer pools, and so on, and tries to adjust their sizes appropriately for the memory size of the computer on which it runs. In practice, this auto-adjustment seldom works well, except on the very smallest servers. One fundamental problem is that if a system has multiple Domino partitions, they operate independently and in ignorance of each other. Each partition sizes its memory use as if it had the whole system to itself! There are two ways to deal with this problem, one that IBM encourages and another that most everybody winds up using instead.

The method IBM recommends is to add “`PercentAvailSysResources=NN`” to each partition's `notes.ini` file, where `NN` is a value between 0 and 100 that indicates the partition's percentage share of the available memory (and some other things, but memory is the important one). The total of the `NN` values for all the partitions should not exceed 100%. For example, on a system running four partitions, you might give each a 25% share, or you might divide the shares unequally if the total is 100% or less, but you should not try to give each of the four partitions a 40% share.

There are at least two problems with this method. First, the value can only specify whole percentages, so on a large-memory system the adjustment might be too coarse: 1% of 64GB is more than 655MB! Second, Domino uses the `NN` value as just one part of a complicated formula, and the formula is tuned for a “least common denominator” operating environment that does not take full advantage of Solaris' memory management policies.

Most people resort to adjusting the `NSF_Buffer_Pool_Size_MB=NNN` parameter, an older and more primitive but tried-and-true method that gives you direct control of the size of the largest and most important of Domino's memory buffer pools. However, the drawback of this second method is that it can be difficult to choose good `NNN` values. One approach that has worked well for me is to begin with about 800MB, and then monitor Domino's performance. This assumes that the machine has sufficient total memory; the total of all the `NNN` values should not exceed about 40 – 45% of the system's physical RAM.

After Domino has been running for a while under load, issue the show stat database console command and study the following three statistics:

- **Database.Database.BufferPool.Maximum.Megabytes** shows the maximum size to which the buffer pool is allowed to grow. This may differ slightly from the *NNN* value you specified, because Domino rounds the size of the pool to fit its own internal data structures, but that's nothing to worry about. If the reported value is very different from your *NNN* value, Domino has probably rejected that value as impossible.
- **Database.Database.BufferPool.Peak.Megabytes** displays the biggest the buffer pool has ever grown since Domino was started. If this is close to the maximum allowed value, Domino is in fact making use of the memory. If it is substantially below the maximum, Domino has not found it necessary to use so much; you have probably given it more memory than it can use and should consider reducing *NNN*.
- **Database.Database.BufferPool.PerCentReadsInBuffer** indicates the effectiveness of the buffer pool. Ideally, you want this value to be 90% or higher. If it is consistently below about 85%, it is likely that Domino is memory-starved and needs a higher *NNN* value.

If you need to enlarge the buffer pool, do so gradually, in steps of 100MB or so. If you get as high as 1,200MB and the *PerCentReadsInBuffer* statistic is still low, it is probably better to split the load across additional partitions than to try to keep increasing the pool size; a pool that gets *too* big can crowd out other essential parts of Domino's memory.

Tuning the file system's memory usage

Now, let's consider the file system's memory. Domino's heavy use of the file system makes its performance important. If the file system has access to plenty of memory, its own caches are that much more effective and the physical disk I/O load are reduced. Not only that, but parallelism within the kernel improves: Certain adjustments to the file system's memory cache require all the system's

hardware strands to cooperate, forcing them to interrupt whatever they are doing and synchronize to keep the file system happy. Reduce the frequency of these adjustments to free up more CPU time for other uses; this is especially important on UltraSPARC T1 and T2 machines, which must synchronize the activities of 32 or 64 "virtual CPUs."

To tune the percentage of physical memory devoted to the file system, edit the */etc/system* file, and at the bottom add the following line:

```
set segmap_percent=NN
```

Save the file and reboot Solaris. The best value for *NN* is difficult or impossible to predict, but I suggest the value of 40% as it should be safe for all but the very smallest systems and is a considerable improvement over the default, which is usually 12%. Too high a value can cause trouble, so don't be tempted to blast up to the exotic values that you'll find in NotesBench reports:¹ Benchmark loads are highly artificial, you should not necessarily accept them as representative of the real world.

How can you tell if you've used too high a value? When the system is under load, run the command *vmstat 10* to display virtual memory statistics every ten seconds (or use some other interval, if you prefer). Find the *sr* (scan rate) column in the output and watch it for a while. All is well if the *sr* value is zero most of the time. An occasional brief spike is nothing to worry about, particularly just as a server or server task is starting, but most of the values should be zero. If they are higher than zero for long periods, it's a sign of trouble, even if the actual values are rather small. Edit */etc/system* again to reduce the *segmap_percent* value a little, and reboot Solaris at the next opportunity.

Use multiple mailboxes

This suggestion aims to eliminate pinch points that could delay activities that would otherwise be nicely parallel. Amdahl's Law teaches us that if we increase

¹ You can find NotesBench reports at www.notesbench.org.

the parallelism of part of a computational problem, eventually the part that cannot be parallelized will limit the attainable throughput. For example, the manager of the local Burger Doodle may hire additional cooks to improve his kitchen's throughput; however, if all the customers must pay at just one cash register, the register's throughput limits the rate at which the restaurant can serve customers. The non-parallelized cash register becomes a pinch point that prevents the restaurant from making full use of its highly parallelized kitchen. To get more productivity from the kitchen, the manager must add parallel cash registers.

A similar pinch point exists in Domino's handling of the mail.box database. This file is the repository for all mail entering its Domino partition, both from local users and from external sources such as other partitions. Server threads deposit incoming mail in mail.box, and router threads take it out and deliver it locally or transfer it toward its ultimate destination. The problem is that only one thread at a time can be actively modifying mail.box. This means that the server and router threads compete with each other to access the database, and threads must wait for access like customers at our analogous burger joint. This pinch point limits the parallelism of the server and reduces Domino's throughput.

Fortunately, it is easy to do the equivalent of installing more cash registers by spreading the load across multiple databases, which reduces the amount of contention for mail.box and relieves the pinch point. Follow these steps to add more mail.box databases and increase parallelism:

1. Start the Domino Administrator Client, connect to the server you wish to tune, and select the Configuration tab.
2. Expand the Server tree in the left-hand panel and select the Configurations view.
3. Open or create the server's Configuration document (or a default Configuration document that applies to multiple servers).
4. Select the Router/SMTP and Basics tabs.
5. In the 'Number of mailboxes' field, enter a number from two to four. Save and close the

Configuration document, and replicate it to the other server partitions it affects, if any.

The next time the partition restarts, it will use mail1.box, mail2.box, and so on instead of just one mail.box.

Note!

You can configure Domino to have up to ten mailbox databases. However, the greatest benefit to parallelism comes from the first few.

You're not quite finished yet, though. When the single-mailbox server partition shuts down, there may be some undelivered messages left in its mail.box database. When the partition restarts and begins using its multiple mail*N*.box files instead, any such messages just sit there and languish. To make sure they are delivered to their intended recipients, open the mail.box file and manually copy any messages you find in it to mail1.box. Once this has been done, you can delete the old and now unused mail.box database. You do not need to repeat this step if you increase the mailbox count again, but if you reduce it, then you need to take similar measures with mail*N*.box files that are no longer used. For example, if you reduce the count from four to two, you should move any stranded messages from mail3.box and mail4.box to one of the active mailboxes; if you reduce the count to one, you should open all of the mail*N*.box files and copy their messages into mail.box.

For more information on this adjustment, see the topics "Creating multiple MAIL.BOX databases" and "Determining how many MAIL.BOX databases to place on a server" in the Domino Administrator help files.

Increase Web concurrency

On a Domino partition that serves Domino Web Access clients or provides other Web services,

consider using more threads in the HTTP task. Your Niagara machine has enough hardware strands to provide for them all. There are two different adjustments that increase Web concurrency; you can make both by editing the partition's Server document:

1. Start the Domino Administrator Client, connect to the server you wish to tune, and select the Configuration tab.
2. Expand the Server tree in the left-hand panel and select the All Server Documents view.
3. Edit the Server document for the partition you wish to tune.
4. For the first adjustment, select the HTTP tab and find the 'Number of active threads' field. For an empty field, the default value is 40; you can improve the concurrency of a busy Web partition by raising this value to 80 or even a little more.

Don't go overboard; IBM strongly discourages exceeding 100 because of memory consumption. There's also the possibility of increasing inter-thread interference; remember the 180 post hole diggers!

The second adjustment allows Web agents to run concurrently instead of serially:

1. Select Internet Protocols → Domino Web Engine.
2. Set the 'Run Web agents concurrently?' field to Enabled.

See the topic "Running Web agents" in the Domino Administrator help files for more information on controlling Web agent execution.

Conclusion

The UltraSPARC T1 and T2 Niagara high-throughput processors represent a new approach to bridging the processor/memory speed gap. From this article, you've learned five tuning suggestions for improving the parallelism of Domino's activities in order to take advantage of the thread-rich environment of the T1 and T2:

- Updating Solaris
- Tuning the Solaris scheduler
- Adjusting Domino's and your file system's use of memory
- Using multiple mailboxes
- Maximizing the parallelism of Web agents and services

These are by no means the only important tuning areas. If you intend to get the most out of Domino, I encourage you to improve your users' experience by taking other measures in addition to tuning for concurrency.

Resources

The Domino Administrator help files contain vital, quick references to INI parameters, settings in various configuration documents, and much more.

The IBM Redbook *Domino 7 for Sun Solaris 10* is a compendium of best practices for sizing, configuring, tuning, monitoring, and troubleshooting Domino on Solaris:
www.redbooks.ibm.com/abstracts/sg247162.html

For a collection of brief tuning opportunities (some are covered in more detail in this article) for Domino on Solaris, refer to *Domino on Solaris: Common Tuning Tips* in the Technical Documentation section of Sun Microsystems' Web site:
www.sun.com/lotus/

THE VIEW • The Sphere

990 Washington Street, Suite 308S

Dedham, MA 02026-6714 USA

phone: +1 781-751-8799 • fax: +1 781-407-9013

www.eVIEW.com • www.SphereJournal.com