



Performance Oriented System Administration

Bob Larson, Strategic Applications Engineering

Sun BluePrints™ OnLine—December 2002



<http://www.sun.com/blueprints>

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-1054-11
Revision 1.0, 1/22/03
Edition: December 2002

Copyright 2002 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95045 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This product or document is protected by copyright and 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 United States and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, Sun Enterprise, Sun Fire, Sun StorEdge, Solstice DiskSuite, Solaris, and Starfire are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US 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.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the Far and its supplements.

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 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95045 Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et 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 enregistrée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

Sun, Sun Microsystems, le logo Sun, Sun BluePrints, Sun Enterprise, Sun Fire, Sun StorEdge, Solstice DiskSuite, Solaris, et Starfire 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'ÉTAT" 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.



Please
Recycle



Adobe PostScript

Performance Oriented System Administration

In most cases, using the default configuration for an operating system (OS) helps ensure that cascading effects don't overly complicate system tuning and maintenance. In some cases, however, you might need to tune a system. When making important trade-offs (for example, of speed over space efficiency), you might need to tune an OS algorithm or data structure.

This Sun BluePrints™ OnLine article explains the algorithms and heuristics surrounding the most important tunables. It describes several kernel tunables and the algorithms behind them.

Note – The *Solaris™ Operating Environment (Solaris OE) Tunable Parameters Reference Manual* available at <http://docs.sun.com> contains detailed information about most of the tunables mentioned in this article.

Tuning for Performance

Because system tuning can have a dramatic effect (either positive or negative) on the performance of a system, carefully consider every change to the default configuration. Whenever you upgrade a system, ensure that each change from the default in `/etc/system` is absolutely essential. Even some kernel patches can change tunable settings, and should be carefully considered.

Further, keep in mind that it is almost always preferable to tune the application rather than to tune the kernel. For example, applications should use `setsockopt(3N)` rather than `ndd(1M)` to set the socket buffer, because using `setsockopt(3N)` allows different applications to set their own values instead of running system defaults.

The following sections provide recommendations for system tuning for performance.

Properly Size Swap

Properly sizing swap can be a trade-off because having too much or too little swap space can cause problems. At a minimum, sizing swap to be too large wastes disk space. A more serious problem, however, is that sizing swap to be too large can cause hard-pageout¹ to occur too easily. The other problem, not allotting enough swap space, can cause the system to run out of virtual memory before it runs out of physical memory. Physical memory is too expensive to waste by not having enough swap disk to use it all.

In addition to balancing these types of trade-offs, you need to be able to recognize when swap is improperly sized. Because neither situation (having too much nor too little swap space) prints a simple message to the console like “Swap too small,” look, instead, for memory allocation errors or somewhat obscure messages such as “cannot fork,” or “not enough memory” as signs that swap is improperly sized. You will never see the message “Swap too large,” even though it can happen, if a machine has a working set size much larger than available memory.

Deciding How Much Space to Add

While it is simple to adjust swap by adding or removing disk partitions or logical volume manager volumes, it is not simple to calculate how much space to add. Your decision depends on the application and on the details of the running environment. Because the application might have tunables to adjust the working set size, it is often better to keep a lid on the requests by keeping the swap size down. This allows the application specialist to quickly recognize a need to resize swap and to change configuration parameters to reduce the working set size without having to look at swap and `vmstat(1M)` outputs.

As memory density grows, the definition of wasted memory needs to grow, as well. You might get to a point when it is not worth struggling to recover larger and larger amounts of memory. When you look at wasted memory, do so with a threshold set in cost rather than in megabytes. What looks like a wasted 100 megabytes of memory, on one hand, can alternatively be viewed as a cheap insurance policy when you consider the potential for it to help you avoid performance problems.

1. Hard-pageout denotes the case where the virtual memory manager has to save active pages onto the swap disk that will have to be re-read in a reasonably short period of time.

Sharing Swap With `/tmp`

A complicating factor for properly sizing swap, especially on larger machines, is the way that `/tmp` is shared with swap. While sharing `/tmp` with swap simplifies administration on smaller machines, it complicates swap sizing because it makes it too easy to burn up swap space in `/tmp` without root privilege. Instead of sharing `/tmp` with swap, make `/tmp` a regular disk. This helps you avoid the performance problems associated with difficulties in properly sizing swap when it is shared with `/tmp`.

Sizing Swap

Because of the implementation of swap in the Solaris OE, the time-honored rule of thumb to size the swap device at two to three times the size of real memory is no longer accurate. These values each need to be reduced by at least one. Sizing swap at one to two times real memory is a good starting point, but even that is too large for systems with very large amounts of physical memory. Depending on the application, very large memory is used only by programs that have higher than traditional ratios of working set size to total size.

Starting with the Solaris 9 OE, you can use the `mdb(1M)` command to analyze the free memory and swap requirements much more easily and accurately than you could in earlier releases of the OE.

The `freemem` statistic, which shows up in `vmstat(1M)` (in the memory free column) and in other monitors, is much more useful for memory sizing in the Solaris 8 OE and later versions than it was in earlier versions. The `freemem` statistic no longer includes memory used by a file that is a valid copy of the on-disk file, and has no processes mapping. These pages are available to be used for new processes on all versions of the Solaris OE kernel. In earlier versions, these pages were considered to be used. In the Solaris 8 OE and later, the free memory statistic doesn't include these pages and, therefore, it more closely matches how big a job can be without sending the system into pageout.

Ignore High `iowait`

The `iowait` statistic can be misleading on large machines. In the past, certain systems required the central processing unit (CPU) that initiated an input/output (I/O) to service the I/O interrupt. On some systems, that meant that the CPU was not available for other processes until the I/O request was satisfied. Because of this situation, I/O wait time came to be considered one of the states of a processor. A processor was either busy in user code, busy in kernel code, sitting in I/O wait, or idle and, therefore, free to run other processes.

In all versions of the Solaris OE, there is no relationship between the CPU that initiates the I/O (for example, through a read or write call) and the CPU that will service the I/O with the interrupt code. As soon as the I/O is initiated, the CPU is free to go back to the run queue and pick up another process, in other words, the CPU is idle. Normally, some other CPU will process the I/O interrupt when it arrives. When that is done, the process waiting on that I/O is awakened. While this change frees the CPU to run other processes, the useful aspect of the old wait I/O statistic, that it allowed you to quantify when idle CPU could be working if the I/O were faster, is now gone.

Determining the Number of Idle CPUs on a Queue

On a multithreaded OS with multiple CPUs and disks, the notion of how many idle CPUs are waiting on a queue of outstanding I/Os is not clearly defined. To answer that question, the system would have to predict the future. For example, if a tape drive had 10 outstanding I/O requests that took 10 minutes to fulfill, just a tiny fraction of a single CPU's time could be used to finish the work for those I/Os. Conceptually, there would be less than one percent of the CPU in `iowait`. On the other hand, a very fast storage subsystem with 10 outstanding I/Os might need a whole CPU to service those requests. Here, on a four CPU machine, 25 percent of the `vmstat(1M)` CPU percentage would be in `iowait`.

We need to look at it another way. There should only be three states for a processor: user, kernel, and idle. Use the device statistics, as reported by `iostat -x`, to determine whether more I/O would release bottlenecks.

On a multiprocessor machine, a single process can do enough asynchronous I/O to make `sar(1M)` show 95 percent I/O wait, making this the apparent condition of the entire system. The value associated with `iowait` is not particularly meaningful for multiprocessor systems because it gives the incorrect impression that one thread issuing I/O requests in a serial manner can tie up all the processors.

There have been attempts to fix `iowait`, but none can overcome the basic problem. On one operating system, a new formula equated a percentage point of `iowait` to each outstanding I/O. This is a confusing work around that simply doesn't recognize the huge range of speeds that need to be dealt with. A large Sun Fire™ 15K server might have 100,000 outstanding I/Os with 50 CPUs idle, and unfortunately, without further analysis, there is no way to tell whether it is I/O-bound or CPU-bound.

Use Logging Mount Options

It is especially important to use logging mount options on any file system where `fsck(1M)` is impractical. Running `fsck(1M)` has limitations in both the time spent and the impact it has in the case of a failure. Failures that occur when running `fsck(1M)` can cause recent changes to files to be lost and can cause whole files to lose their names by being saved in lost and found.

Logging file systems treat metadata updates as atomic operations with an intent, operation, and confirm sequence that makes the consistency check trivial, and, therefore, take little time.

For file systems other than root, you can change this option dynamically without rebooting or unmounting. Use the `mount(1M)` command's `remount` option and `log /mnt` as follows:

```
mount -o remount
```

Then, make the option change in `/etc/vfstab` to automatically pick up this option upon reboot.

Boost maxphys

Beginning in Solaris OE version 2.5.1, large I/Os to disks and logical volumes are possible (raw and direct mount UNIX® file system [UFS] only). The default size of 128 kilobytes is not large enough for large systems. The recommendation is to increase the size to 1 megabyte, although many sites use even higher values (typically 16 or 4 megabytes).

When using logical volume managers, additional variables need to track the increase in `maxphys` as follows:

- Increase `maxphys: vxio:vol_maxio` listed in disk blocks or in half of a kilobyte.
- Increase `md_maxphys` listed in bytes.

Increase Blocks Per Inode in Big File Systems

Because UNIX file systems are general purpose file systems, many trade-offs are made when they are initialized. For large systems running databases, you can improve performance by tuning `newfs(1M)` parameters. The default for the average number of bytes per inode (`nbpi`) in the file system is statically set for older versions

of the Solaris OE. The Solaris 8 OE `newfs(1M)` command has a simple heuristic: it chooses a size for the new file system between 2048 and 8192 bytes. This assumes that the average file size is around 2 kilobytes to 8 kilobytes.

By default, the `newfs(1M)` command optimizes for a file system with capacity demands that grow and shrink regularly. This profile is usually inappropriate when creating a file system for a database server. In a database, the average file size will be much larger than 2 to 8 kilobytes. Change the `nbpi` to 500 kilobytes or higher with the `-i` option to `newfs(1M)`. This greatly reduces the time required to complete the `newfs(1M)` command. On the other hand, some systems use large numbers of symbolic links, which use an inode with zero bytes. So the bytes per inode might need to be even lower than the default value.

In Solaris versions 2.6 and earlier, besides `maxcontig` (discussed in a later section), the minimum free percentage and cylinders per cylinder group can also help you reclaim about 15 percent of the disk space. For large files, and for large disks, you can increase the cylinders per cylinder group to the largest allowable value, around 400 for Solaris up to Solaris 9, and 251 for Solaris 9. When set with the `newfs(1M) -c` option, this variable is best set to the maximum for large filesystems with few files. In Solaris OE versions before 7, by default, the minimum free percentage is set to 10 percent. Use the `newfs(1M) -m` option to change the default value. When a file system is devoted to a database, there is not enough dynamic space allocated to make 10 percent necessary and you can set the value to 2 percent or less. The combination of the `-m` and `-i` options will save about 15 percent of the total space, or 150 gigabytes on a one terabyte file system.

Several options to `mkfs(1M)` and `newfs(1M)` are not used. They are still documented, but have no affect on modern disks. Some of the parameters that are ignored are `rotdelay` (or `gap`) and `rpm`.

Tune `fsflush`

`fsflush` is a kernel thread that performs several tasks including a cyclical scan of all pages in memory. In systems with large quantities of physical memory, `fsflush` might become overloaded. `fsflush` can consume excessive CPU cycles because it must complete a scan in a small, fixed amount of time, as defined by the tunable variable called `autoup`. Use the variable `tune_t_fsflushr` to define the interval of time, in seconds, for `fsflush` to wake up to check buffers. Set these variables in `/etc/system`.

When `fsflush` scans pages, it moves writable `mmap` pages that are dirty out to the disk. This prepares the system for pageout and helps the system recover used memory. Creating free memory is especially important when many processes are consuming memory through `malloc(3C)` or `read(2)` or `mmap(2)`.

The default of 30 seconds for `autoup` makes the `fsflush` process too costly on a big-memory system. To cool off `fsflush`, increase the cycle time by increasing the `autoup` value. This will hit diminishing returns above a few minutes.

The effect of `fsflush` is system wide, so it is best to make it run for short periods of time by decreasing `tune_t_fsflushr`, the time between wakeups, to one second from the default of five seconds. This makes the `fsflush` process do one fifth as much work per wakeup. Decreasing `fsflush` improves performance even though it wakes up five times more often than it does with the default settings.

Tune `maxcontig`

Under the Solaris OE, UFS uses an extent-like feature called clustering. It is impossible to have a default setting for `maxcontig` that is optimal for all file systems. It is too application dependent. Many small files accessed in a random pattern do not need extents and performance can suffer for both reads and writes when using extents. Larger files can benefit from read-ahead on reads and improved allocation units when using extents in writes.

For reads, the extent-like feature is really just a read ahead. To simply and dynamically tune the read-ahead algorithm, use the `tunefs(1M)` command as follows:

```
# tunefs -a 4 /ufs1
```

The value changed is `maxcontig`, which sets the number of file system blocks read in read ahead. The preceding example changes the maximum contiguous block count from 32 (the default) to 4.

When a process reads more than one file system block, the kernel schedules reads to fill the rest of `maxcontig * file system blocksize` bytes. A single 8 kilobyte, or smaller, random read on a file does not trigger read ahead. Read ahead does not occur on files being read with `mmap(2)`.

The kernel attempts to automatically detect whether an application is doing small random or large sequential I/O. This often works fine, but the definition of small or large depends more on system application criteria than on device characteristics. Tune `maxcontig` to obtain optimal performance.

With the Solaris 2.5.1 OE, the default for `maxcontig` was always 7. After that, the default changed to be a device-specific value. When a logical volume manager or disk striping package is used, such as Solstice DiskSuite™ software, the default of the software overrides the device value. The following values apply for the specified devices:

TABLE 1 Device-Specific Values for `maxcontig`

Device	Default <code>maxcontig</code>
<code>sd</code>	<code>maxphys</code>
<code>ssd</code>	1 megabyte
VERITAS Volume Manager (VxVM)	<code>stripe_width 512 K</code>
Solstice DiskSuite software	<code>maxphys</code>

Because hardware with redundant array of independent disk (RAID) devices (like the Sun StorEdge™ T3) read ahead internally, the benefit of file system layer read ahead is diminished. For devices with RAID, lower `maxcontig` values reduce memory pressure.

When running a transaction processing database, a database will not usually use any read-ahead blocks. Therefore, all blocks pulled in for the read ahead just add wasted activity on the disks. In this situation, set `maxcontig` to 1.

Conversely, for large files accessed in a roughly sequential manner, a boosted `maxcontig` with a boosted `maxphys` can improve transfer efficiency by increasing the amount of data read once disk heads are positioned within a cylinder. The best value is to have the read ahead as large as a cylinder, around 1 megabyte, which translates to a `maxcontig` of 128 bytes.

Base choices about the values for `maxcontig` on the likelihood of getting good data out of blocks that are near each other on a disk (or volume) versus the wasted I/O and the memory that holds those blocks if they are never used.

Set `noatime` on mount

Some applications, especially databases, don't need the OS to maintain the most recent access time that is stored in the file system metadata called `atime`. Also, mail and messaging systems have very active file systems where the most recently accessed time is not especially important. By default, the `mount` option enables access time updates, which means reads to the file have to eventually write this metadata with a new timestamp. This can be detrimental to performance.

In versions of the Solaris OE from version 7 forward, use the `-noatime` option for the `mount` command to relax the update policy on access time.

Tune the Network

Network tuning can be quite different for a machine that is doing large transfers over persistent connections than it is for one with HTTP style traffic. To ensure that the system has plenty of sockets available, HTTP servers commonly adjust the `tcp_time_wait_interval` parameter.

It is common to need an adjustment of the `tcp_conn_hash_size` parameter, which will help the machine with a large number of connections. For bandwidth intensive servers, the `xmit_hiwat` and `recv_hiwat` parameters probably deserve attention, especially on older versions, but most of these servers have defaults that are large enough.

Don't let the machine accidentally run as a router. Using more than one network interface card (NIC) will, by default, turn on routing in the kernel between those two subnets. By simply using the file `/etc/notrouter` as follows, the initialization scripts don't let this happen.

```
touch /etc/notrouter
```

When applications attempt to increase their networking buffer sizes by calling `setsockopt(3N)` explicitly, the system checks the request against the parameters `udp_max_buf` and `tcp_max_buf`. These parameters can be adjusted to allow larger than the defaults with `ndd`. The defaults for the parameters are probably large enough for most applications, at 1048576 for `tcp` and 262144 for `udp`, but for bandwidth intensive applications, larger values are useful.

For latency sensitive applications, the default settings make some transmissions wait to attempt to fill a packet, which kills lock-manager and latency-sensitive query performance. To fix this situation, set `TCP_NODELAY` in the system call `setsockopt(3N)`. Most databases and sophisticated network applications have an option to do this. Here, the default is for throughput rather than for latency.

Dispatch Table Tuning

The dispatch table is used by the kernel scheduler to make choices about which process runs when CPUs are too busy for every process to have exclusive use of a CPU. From the classic point of view, CPU-intensive processes are pushed to lower priority automatically. This is achieved by noting that the process is still running when the time quantum at their level expires. Interactive performance is enhanced

by letting the associated processes float to a higher priority by noting that it has gone to sleep (waiting for I/O) before their quanta expires. This classic view is how the default Sun timeshare dispatch table works.

On the other hand, for big database servers, this action can cause a CPU-intensive portion of the database to hold a data structure lock as it runs at low priority. To avoid this situation, we use higher values for the time quanta, known as the Starfire™ dispatch table or the Cray dispatch table. It is automatically used in certain cases on the Sun Enterprise™10000 server and Sun Fire 15K server and might be useful on other machines as well. This feature is implemented in a slightly different way on different versions of the operating environment:

- In the Solaris 2.5.1 and 2.6 OEs, the Sun Enterprise 10000 server loads the big quanta table by default.
- In the Solaris 2.7 update 5/99 OE, this server has a special initialization script that loads on boot called `S99bigquanta`.
- In the Solaris 8 and 9 OE kernels, a platform specific module is loaded, but a script can also be used.

Any user can see what the quanta are by executing the `dispadmin` command as follows:

```
dispadmin -c TS -g
```

The big quanta tables run from 400 milliseconds to 340 milliseconds, while the normal quanta run from 200 milliseconds to 20 milliseconds.

A good way to implement this is to make a new `rc3.d` script to switch back and forth between different quanta based on the stop and start argument. You can do this through `cron` for a system using batch at night and interactive during the day.

Another parameter related to scheduling, `rechoose_interval`, sets a bias on how often the kernel moves threads when unevenly balanced CPU run queues occur. If a process sitting in a run queue waits for `rechoose_interval` clock ticks, it will be moved to another CPU. Do not change this setting unless load balancing is a problem and the machine has long-running, CPU-bound jobs (little-to-no I/O).

Use Fast-Write Disks Appropriately

Database log files are the most important place for fast, reliable writes. As databases become larger, pressure on the log devices can become an important issue. Most databases hold up transactions while the log I/O is in progress. If the log is going to plain mirrored devices, the maximum throughput includes considerable wait time, as the disk rotates to the heads. By using storage devices with fast write caches, you can greatly reduce the time of this wait.

Use fast write enabled disks whenever the read-write balance is biased heavily towards writes. For example, temp files and the system dump device should be fast write, if possible. This is especially good to do when using a UFS mounted disk for `/tmp`.

Look at Console

When troubleshooting hardware problems, make sure to check the console. Many drivers post error information through the `syslogd`. For example, the error log delivery subsystem is managed through the `syslogd` process. It is important to take care of the `syslogd` process and to test and verify it regularly. For example, any time certain known events occur, check the logs to verify that `syslogd` is working.

In addition to the default behavior of sending messages to the log file, to `/var/adm/messages`, and to the console, you can also route messages based on their severity.

Use the Fastest Available File System

The `tempfs` file system type has been optimized for small file and metadata-intensive access. The performance of this file system is not the best that is available for creating large sequential files. A regular UFS mount for `/tmp` is better for working with large files. This is an especially important consideration for sparse file creation because sparse file behavior is emulated by creating the whole file on `tempfs`.

In the Solaris 9 OE, the performance of the `tempfs` file system has been improved. Still, it is desirable to make `/tmp` a regular file system due to its simplification of swap sizing.

Tune for Heavy UFS Writes

To keep slow disks from being overloaded, the kernel blocks processes after a certain number of processes are already found to be waiting. The number of outstanding blocks is specified by the configurable kernel variable `ufs_HW`. If more processes try to write than are specified by `ufs_HW`, the kernel blocks them and increments the `ufs_throttles` counter. The default for `ufs_HW` in versions before the Solaris 9 OE is quite low and should be increased for systems that have high-speed I/O devices underlying the UFS. The counter `ufs_throttles` increments each time the system

throttles on this condition. If `ufs_throttles` is more than a few hundred per day, consider increasing `ufs_HW`. To check `ufs_throttles`, use a kernel debugger as follows:

```
adb -k <<EOT
ufs_throttles/D
EOT
physmem 5dbd6a
ufs_throttles:
ufs_throttles: 0
```

The kernel blocks processes until the disk has worked its queue down to `ufs_LW` (low water). When setting these values, ensure that `ufs_LW` and `ufs_HW` work together. Ensure that `ufs_HW` is always higher than `ufs_LW`. In the Solaris 9 OE, the default is 16 megabytes for `ufs_HW` and 8 megabytes for `ufs_LW`. These settings would be fine for versions preceding version 9.

In the Solaris 9 OE, make sure that `/etc/system` does not have a setting of `ufs_HW= 1 megabyte`. Having a higher LW than HW will hang the system.

Manage File System Fragmentation in UNIX

The UFS can have problems with fragmentation when multiple files are created on the same file system at the same time. Extent-based file systems solve this with additional parameters that create a reserved area when the first few blocks are written so that another concurrent writer will not get blocks immediately after the first blocks are written. In UFS, the effective extent size is quite small: 64 kilobytes.

To avoid fragmentation, don't allow multiple writers in one file system, especially when large quasi-permanent files are being created, namely during database table space creation.

To eliminate fragmentation, use `ufsdump` and `ufsrestore` smiley pipe to do a disk-to-disk dump restore as follows:

```
ufsdump 0f - . | ( cd dest_dir && \
ufsrestore xf -)
```

For example, on a small file system, a comparison of one case where write fragmentation is allowed and another case where write fragmentation is avoided shows considerably better results when doing sequential reads.

Tune for Heavy UFS Read

UFS read tuning normally has several goals, depending on the application level access patterns. Reusing buffered files either contributes to efficiency or just adds to memory use. When a large set of files is read in a processing step, often no reuse will be seen within the memory size constraints of the server. This case is typically recognized by sequentially accessing large files, and the goal will be to keep these accesses from polluting file system caches. On the other hand, a random read workload from files of any size is expected to have access patterns that fall in a distribution where a good percentage will get the efficiency gain of a cache hit. In this situation, the goal is to maximize cache hits. In either case, you should leave enough memory for programs to have their executable and libraries in memory without pageout.

Several kernel variables allow you to tune the thresholds where the different cache management policies are invoked. In the Solaris 9 OE, the variable `segmap_percent` allows you to set the percentage of physical memory to devote to file buffering. The default is 12 percent, leaving 88 percent of memory for executables. Picking a site-specific number for this value is an important step in overall system tuning.

The algorithm to use to avoid pollution is controlled by `smallfile` and `freebehind`. By setting `freebehind` to `off`, (the default is `on`) the system attempts to save the contents of large sequentially accessed files in the buffer cache. By leaving this option on and setting `smallfile` to a higher value, the system can cache larger files. The default value of `smallfile` is only 32,768 bytes. On a large server where 2-gigabyte data files are being used, there is nothing wrong with setting `smallfile` to 2,147,483,647 bytes if heavy use of the buffer cache is desirable.

Before version 8 of the Solaris OE kernel, it was quite common for the machine to detect a memory shortfall during heavy concurrent sequential reads. Then, the kernel attempts to keep the memory use of file system buffers to a minimum with the following rule. If the file is larger than the dynamically changeable kernel variable `small file`, then the kernel changes from its normal least recently used (LRU) buffer replacement policy to most recently used (MRU) replacement when doing writes to that file.

In the Solaris 8 OE, the definition of memory shortfall is quite different than it is for the Solaris 7 OE and earlier versions. In the Solaris 8 OE, the `freebehind` is less aggressive.

Sometimes the MRU algorithm on large files is exactly what the system needs. Other times, `freebehind` gets in the way.

When the application could be reused, but isn't, `freebehind` evicts buffers based on an incorrect assumption that they will not be reused. For example, in the following sort function, the second read (random read in target order) will have very bad performance if the file is not kept in the cache during the index sort phase, having been read during the first read.

```
sequential read
build index
in-memory sort based on index
create new ordering index
    in target output order
random read in target order
sequential write
```

When the system has plenty of memory, the file system doesn't do `freebehind`, and all those blocks will be hits in the page cache. When the system is in memory shortfall (maybe because a lot of jobs are running in parallel), the `freebehind` algorithm will not allow the first read to save pages in the page cache. This is a problem because the algorithm will save the pages from the second read to the page cache. While this might seem like a bug, it's really a case where the heuristic designed for the general case has a pathological counter case.

In this case and in the case of heavy UFS writes, there might appear to be a lack of scalability, but tuning the single-way job doesn't finish the job of tuning. The system will not hit the conditions that limit performance when all the CPUs are used. Some additional work has to be done to tune the concurrent case. The characteristics of the job changes, particularly with respect to memory pressure. Often, even if a single user or job has a sequential I/O pattern and no issues with memory shortfall, the characteristics reverse when enough jobs are launched to use all the CPUs.

Size IPC Semaphore Properly

Normally, the application vendor's recommendations are fine for interprocess communication (IPC) sizing. It is normally fine to simply add up the recommendations from each application being installed on a per-parameter basis. However, the undo structures are sometimes over configured when doing this.

Using large values for System V shared memory with semaphore configuration parameters produces overhead by increasing internal tables to hold more semaphore identifiers and especially semaphore undo structures. Large values for the system-wide semaphore-limit variable, `semmsl`, use enough memory to make it worthwhile to size the variable properly. The product of `semmsl` (the maximum number of

semaphores per set) and `semnmi` (the system-wide maximum number of semaphore sets) is a second maximum, so by maintaining the relationship as follows, memory won't be wasted:

```
semnms = semmsl * semnmi
```

The undo structure gives the application a chance to make calls to `semop(2)` with the `SEM_UNDO` flag set, and to back out the semaphore operation in the event that the calling process is killed or dies on an error condition. To do this quickly, a small structure is allocated for each `semop(2)` that uses the `SEM_UNDO` flag. The product of `seminfo_semmnu * seminfo_semume * 8` (bytes) will be allocated when a process calls `semop` with the `SEM_UNDO` flag. Seemingly small values for these variables turn into too much memory quickly.

Other Variables to Optimize Memory Use

Two variables often need to be adjusted on large memory machines that will result in wasted memory in most cases if not tuned. Because these variables are based on a percentage of physical memory installed, they are only important on domains with more than 50 gigabytes, or so. These commands are `segspt_minfree`, which uses 5 percent of memory, and `bufhwm`, which uses 2 percent of memory by default. Both of these commands can be fixed at generous values that are much smaller than their default values. The recommendation is to set `bufhwm` to 8 megabytes and to set `segspt_minfree` to 80 megabytes.

In addition, you can save kernel memory on large machines by decreasing the value of the `maxusers` variable. This variable is set through a heuristic that makes sense for most installations, but if a large memory machine doesn't need a large number of users or a large number of processes, you can reduce it to save memory.

In addition, it is best to disable the `shminfo_shmmax` variable altogether. It is not an effective way to limit memory use and, therefore, is normally set to unlimited as follows:

```
set shmsys:shminfo_shmmax=0xffffffffffff
```

Delay savecore After Panic

Because of the activity early in the boot to do the `savecore`, the memory freelist might become fragmented. This can prevent the shared memory from getting full allocation in 4-megabyte pages. To avoid this problem, you can start the database before the `savecore` or you can `umount` or `direct mount` the file system where the crash core files are kept.

Keep Copies of Crucial Files

Teamwork is enhanced when system administrators can easily find an online history of system configuration files such as partition tables, logical volume manager layout files, and the like.

Consider the size of some files versus the critical nature of customizations. Keep spare or pre-modification versions of `/etc/system`, partition maps of disks, `vfstab`, and the like on the server in the `/etc` directory. Several methods for doing this are plausible. At a minimum, save current configurations often with a simple alias as follows:

```
alias newv='mv /etc/vx.out /etc/vx.out.old; \  
date > /etc/vx.out ; \  
vxprint -thr >> /etc/vx.out'  
format << EOT > /etc/disks.format.out  
EOT  
prtvtoc /dev/rdisk/clt0d0s2 \  
> /etc/vtoc.ddisk
```

Manage Core Files

When problems occur in an application, the default action is for a core image to be saved in the current directory. This can be a large file that is only rarely used. The recommendation is to reverse the default action by using `limit coredumpsize 0` in `/etc/.login` and `ulimit -c 0` in `/etc/profile` for the Solaris OE, version 7 and earlier.

In the Solaris 8 OE, the system policy is more easily managed with the `coreadm(1M)` utility.

Avoid Rebooting for Mount Issues

Disk failures can lead to problems with `mount` and `umount`, but don't necessarily require reboots. If a scratch or unmirrored disk fails, you can do a `umount` with the `umount -f` command. If a mount fails after using the `umount -f` command, reinitialize a structure in the kernel by simply removing the mount point and making a new one as follows:

```
rmdir mntpoint
mkdir mntpoint
```

Use Alternatives to `mkfile`

The blocksize `mkfile(1M)` uses 8192 bytes. This is too small for evaluating large file creation performance without the help of the `maxcontig` command. This test of large file creation time implicitly uses a small blocksize, making it inappropriate for direct mounts.

```
timex mkfile 100m \
/t1/swapfile-100m
```

With no option to increase the blocksize in `mkfile`, use a different utility to do the same test. For example, `dd(1M)` accepts a `bs=` parameter to use a 1 megabyte blocksize as shown here:

```
dd if=/dev/zero of=/t1/swapfile-100m \
bs=1024k count=100
```

Another solution is to temporarily change `maxcontig` to 32 megabytes, or more, when using programs that write sequential data in small blocks.

Install and Use a Permanent Recorder of Overall System Performance

To facilitate performance management and to track and predict load over time, extend `iostat` or other simple utilities to generate permanent records of performance and accounting information. In its simplest form, all this requires is for you to run a simple `cron` job as follows:

```
0 0 * * * ( IFS="";trap 'kill 0' 0; vmstat 2|&
while read -p line; do X=`date +%Y.%m.%d.%
H:%M:%S`; echo $X $line; done) >>
/var//vmstat.`date +%m%d`
```

Of course the `sar` command or even commercial products such as Teamquest or BMC, can do this, and the simple tracking and logging suggested here is in no way a replacement for these more sophisticated tools. The simple loggers, however, have some advantages in flexibility of the data they generate.

Other Miscellaneous Tips

The following suggestions can help you avoid some common problems.

Performing Full Installations to Avoid Future Problems

With the size of modern disks, always do a full installation on servers. There's no point in trying to save a few hundred megabytes to later find out that you need a package that's only installed on the server option.

Finding Package Commands or Files

Use the plain text file `/var/sadm/install/contents` to be comfortable with package-name-to-utility-name relationships. Using this file makes it easy to answer the question “What is the package name for Perl?” To use this file, simply type the commands that follow:

```
grep '/usr/bin/perl' \  
/var/sadm/install/contents  
/usr/bin/perl=../perl5/5.00503  
/bin/perl5.00503 1 none SUNWp15u
```

Defining Man Page Printing Options

To ensure that you can print a nicely formatted PostScript file of man pages, enabling you to build up a personal reference pack, set the environment variable `TCAT` as follows:

```
TCAT=usr/lib/lp/postscript/dpost  
man -t whatever | lp
```

About the Author

Bob Larson concentrates on overall system performance within a benchmarking group that has access to the largest and newest machines that Sun makes. The tasks Bob performs include performance planning and sizing, and setting up system architectures with detailed experiments and prototyping. Bob has over 12 years experience in performance tuning, application and kernel development, and benchmarking, and has published many world-record, industry-standard benchmarks including SAP, Peoplesoft, Oracle, DB/2, Sybase, TPC, and SPEC.

Bob has written articles and papers that document ways to recognize performance problems and tune the OE and databases. Recently, his article “Tuning Large Solaris Servers” appeared in *SysAdmin Magazine*. Bob pioneered several performance tuning techniques that are practical and significant: Dispatch table tuning, wide-thin-striping, interrupt pinning and fencing, and floater board performance characterization. Bob has authored low-level disk testing utilities used widely in sizing and characterization of system I/O performance.

Bob holds a Bachelor's degree in Physics from the University of Chicago.

Acknowledgements

This article would not have been possible without the help of my Sun Strategic Applications Engineering colleagues.

References

Solaris™ Operating Environment (Solaris OE) Tunable Parameters Reference Manual.

Ordering Sun Documents

The SunDocsSM program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`