



ORACLE for Sun

Performance Tuning Tips

Oracle for Sun
Performance Tuning Tips

Part No. A25584-1
February 1995

Author: Rajesh Ragoobeer
Contributors: Henry Dai, Mike Sakayeda, Karl Haas, Peter Lai, Paul Turner, Cindy Lim, Andrew Holdsworth, Carlos Godinez, Don Tirsell, Allan Packer, Matthew Kagle

Copyright (c) Oracle Coporation 1995
All rights reserved. Printed in the U.S.A.

Restricted Rights Legend

Use, duplication, or disclosure is subject to restrictions stated in your contract with Oracle Corporation.

Use, duplication, or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed Restricted Rights software under Federal Law.

The information in this document is subject to change without notice. If you find any problems in this document, please report them to us in writing. Oracle Corporation does not warrant that this document is free of errors.

ORACLE, Oracle*Mail, Oracle*Financials, SQL*DBA, SQL*Plus are registered trademarks of Oracle Corporation.

ORACLE7, Optimal Flexible Architecture, OFA, OFA Standard, Oracle*APS are trademarks of Oracle Corporation.

UNIX is a registered trademark in the United States and other countries of Novell, Inc. licensed exclusively through X/Open Company Limited.

All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

Oracle for Sun Performance Tuning Tips

I. Introduction

Purpose

The purpose of this guide is to help Sun users improve the performance and scalability of Oracle7 database server on Sun/Solaris systems.

This guide needs to be used in conjunction with the *Oracle for UNIX Performance Tuning Tips* that covers generic UNIX tuning tips and other generic Oracle documentation that deals with Oracle tuning issues. This document *expands on those tuning tips that require further Solaris specific information* related to the use of Oracle7 on the Sun/Solaris 2 platforms.

Readers are also recommended to first consult the generic Oracle and Solaris tuning documentation before referring to this document. See *Useful References* at the end of this document.

Using this Guide

This guide illustrates how to use features within Oracle7 products and the Sun/Solaris operating systems to tune the Oracle7 database server and applications. It supersedes previous version of *Oracle for Sun Performance Tuning Tips* (Part# A11357) documentation. The guide has been divided into the following main categories:

- Disk I/O issues
- CPU Scheduling and Process Priorities
- Memory and Paging
- Network / IPC tuning

The reader should note that there is no particular purpose in the order in which the topics are discussed, except for readability.

The amount of effort you should put into tuning of each of the above areas with respect to your environment depends on your application and workload characteristics. However, in our experience the biggest gains (apart from application redesigning and SQL tuning) are achieved by reducing disk and network I/O.

Feedback

The tuning information contained in this documentation is largely the result of the joint performance work between Oracle Corporation and Sun Microsystems Inc. The performance work is ongoing and this documentation will be updated periodically to reflect the new findings. In addition, personal experiences of Oracle and Sun performance specialists also contribute to this guide. Oracle Corporation is very interested in your tuning experiences that will contribute towards improving this guide. We are also interested in your problems, concerns and suggestions. Please provide us your feedback on the effectiveness of this guide. See section on *Feedback* at the end of this document for more details.

II. Disk I/O issues

Tip No. 1: Use Asynchronous I/O

Oracle7 takes full advantages of asynchronous I/O (AIO) provided by Solaris 2 for faster database access. The block diagram (Figure 1) below illustrates the AIO and the Logical Volume Manager (LVM) on the Sun/Solaris platform. AIO interleaves multiple I/O operations to improve I/O subsystem throughput. LVM reduces disk contention by striping data across multiple disk spindles (see section on *Use a Logical Volume Manager* for more details). AIO used together with LVM significantly improves RDBMS performance.

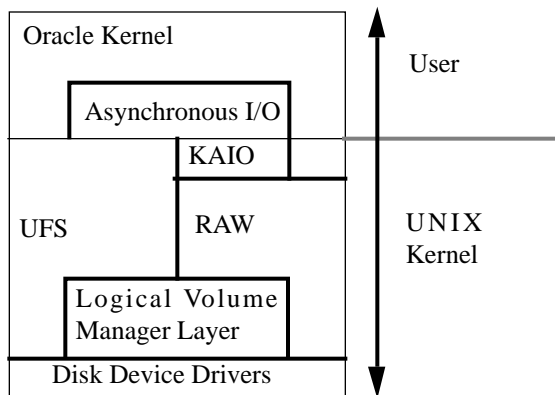


Figure 1: Asynchronous I/O in Solaris 2.4

On Solaris 2, AIO is enabled by default. AIO is supported for database files created on either file system partitions or raw devices.

NOTE: On Solaris 2, AIO for cooked or UFS (UNIX File System) files are done by employing threads to parallelize the disk requests. Refer to the section *CPU Scheduling and Process Priorities* on the implications of using AIO with UFS based datafiles.

With Solaris 2.4, AIO on raw devices is further enhanced by additional technology called KAIO (kernelized AIO). KAIO improves performance by reducing the context switch overhead. This added benefit is transparent to database users and requires *no changes* in Solaris or Oracle configuration. KAIO is supported on Solaris 2.3 via a patch obtainable from Sun.

KAIO on raw devices accompanied by LVM yields the best overall DBMS performance and scalability.

Asynchronous Write

Oracle7 uses asynchronous write for parallel database writes. This feature is turned on by default in ORACLE (i.e. the init.ora parameter **async_write = TRUE**).

Depending on your initial configuration, your database file distribution, your system resource limitations, your overall system load, and most importantly, the nature of the workload, the performance benefit that you might see is roughly 0 - 50%.

Asynchronous Read

Oracle7 for Solaris 2 uses asynchronous read for database recovery. During recovery, ORACLE speeds up the process by issuing parallel reads of database files. Asynchronous read is enabled by default in ORACLE, i.e. the init.ora parameter **async_read = TRUE**.

NOTE: In Oracle 7.1, additional performance gains can be achieved by setting the init.ora parameter **recovery_parallelism = N** (where *N* is an integer greater than 1; usually set to the number of processors on the machine). Here multiple slave recovery processes are spawned to partition the task of applying the logfile changes to the database.

Depending on your database configuration, your system resource limitations, and your overall system load, the performance benefit (for database recovery only) that you might see is roughly 0 - 50%.

Tip No. 2: Use a Logical Volume Manager (LVM)

LVM provides the capability of striping data across multiple disk spindles to reduce disk contention. It is also transparent to ORACLE. Generally, in most environments one cannot exactly determine the characteristics a priori. Thus by using an LVM effectively one can spread the I/O more randomly, resulting in greater overall performance gains. We recommend using the LVM (also referred to as machine striping) over manual striping (striping without using LVM) for almost all workloads. The following logical volume manager products are currently available on Solaris 2:

- Online DiskSuite
- Veritas Volume Manager

The stripe size or interleave value or interlace factor is important to RDBMS performance. It is heavily dependent on the nature of disk access i.e. the I/O characteristics of your workload. Our initial experiments have shown that stripe sizes of 32K and 64K are good values for most workloads. More work is currently being done to better estimate the stripe size for different I/O activity patterns with different workloads.

NOTE: If the stripe size is not specified, it defaults to the cylinder size of the first disk encountered. This may be in the order of a megabyte on some disks, and may hurt ORACLE performance in some cases (especially workloads involving OLTP activity) since RDBMS I/O requests are typically smaller.

There are some issues to consider with LVM. First, avoid using partitions from the same physical disk if these partitions are likely to be accessed simultaneously. For example, since the datafiles and logfiles are accessed simultaneously most of the time, datafiles and logfiles should be striped on different sets of disks. Otherwise, this will result in worse performance due to the excessive disk-head seek activity. This degradation is even worse if the partitions on a disk are far apart from each other.

Second, for capacity planning, use partitions of uniform size when creating a logical/virtual disk. Using disk partitions of different sizes wastes disk space, since the sizes of the partitions in the logical disk is limited by the smallest partition.

Third, use additional LVM features to minimize the impact of the reduced effective mean time between failures (i.e. increased likelihood of failure) of a virtual disk. These high availability features include:

- mirroring or using higher RAID levels on logical/virtual disks
- using the filesystem/volume logging facilities for faster recovery

Please refer to the vendor LVM documentation for the latest information on performance tuning of the LVM.

Fourth, make sure that the I/O activity is evenly dispersed across multiple disk spindles by using Solaris utilities like *sar*, *iostat* or other performance tools to identify any hot disks.

Performance gains from efficiently using the LVM can vary greatly depending on the LVM used and the characteristics of the workload. For decision support system (DSS) workloads we have experienced excellent scalability (0-500%). For OLTP type workloads or mixed workloads one can still expect significant performance gains ranging from a few percent to over 100%.

Tip No. 3:Using readv()

The *readv()* system call increases I/O throughput for sequential read activity by reducing the CPU overhead associated with buffer copying. On Solaris 2, we have observed good performance improvement when using *readv()* on datafiles created on UFS.

However, our experiments on Solaris 2 have revealed that *readv()* degrades performance when database files are created on *raw partitions*. We are currently working with Sun engineers to fix this problem. Therefore, *readv()* should not be used on databases where there are datafiles created on raw partitions.

By default *readv()* is not used by ORACLE i.e. the *init.ora* parameter ***use_readv=FALSE***. To use *readv()* set the *init.ora* parameter ***use_readv=TRUE***.

The expected performance gain when using *readv()* with UFS based datafiles is 10-20%. The performance degradation from using *readv()* on raw devices is observed to be 30-50%.

Tip No. 4:Use db_file_multiblock_read_count

The *init.ora* parameter ***db_file_multiblock_read_count*** usually yields better I/O throughput for large values. On Solaris 2, this parameter ranges from 1 to 512, but changing this beyond 32 usually does not provide any additional performance gain.

This parameter should be set so that *db_block_size* * *db_file_multiblock_read_count* is greater than the LVM stripe size. This causes more disk spindles to be employed in satisfying an I/O request.

Tip No. 5:Using Raid Capabilities

RAID-5 enhances sequential read performance, but degrades overall write performance. Additional work is currently being done to more precisely quantify this performance gain.

Tip No. 6:Disk Geometry Considerations

The use of ZBR (zone bit recording) technology from Sun causes the outermost cylinders to be more efficiently utilized. This results in better overall performance when the outer cylinders are used compared to the inner cylinders.

Some users have seen 0-10% increase in I/O throughput for large I/O requests.

Tip No. 7:Using UFS vs. Raw Partitions

This topic generates a fair amount of debate from time to time. Some of the reasons for this include the following:

- Filesystems are continually being improved and this creates a moving target that makes performance analysis very difficult. So while disk technologies are improving, so do the various filesystem implementations. This results in some cases where filesystems provide better I/O performance than raw devices.
- Different vendors implement the filesystem layer in their own innovative ways to exploit the strengths of different disk peripherals in various ways. This further complicates the users perception across platforms.
- With the introduction of more powerful LVM interfaces, the burden of configuring and backing logical disks based on raw partitions are eased substantially.

The performance experienced on depends greatly on the I/O characteristics of the workload.

Advantages of UFS:

From the Sun perspective we see that UFS is easier to use from an operational viewpoint compared to raw devices. However, as indicated above LVM eases the administration of raw devices.

On lightly loaded systems, where there isn't heavy contention for UNIX buffers, sequential reads (especially repeated reads) benefit from UFS substantially. The reasons for this is twofold:

- 1) Presumably, filesystem read-ahead capability kicks in when it detects large sequential read accesses.
- 2) The data ends up in the UNIX kernel buffer cache which speeds up subsequent scans on the same object.

3) The logical block size for the filesystem is configured when you create the filesystem. This can be chosen to match with the Oracle database blocksize.

Another advantage is the opportunity to use the Solaris system call `readv()` which boosts sequential read performance. See earlier section on *Using readv()* for more detailed information.

Disadvantages of UFS:

The main downside to using UFS based datafiles is the extra buffer copies that need to take place between the Oracle buffers cache (SGA) and the UNIX buffer cache, every time data is transferred to and from the I/O subsystem. This penalty becomes even greater when the I/O subsystem is more heavily exercised.

Another major disadvantage becomes apparent on heavily loaded systems where contention for UNIX buffers end up becoming the main bottleneck. Some side-effects include:

1. More work to be done by the pagedaemon that flushes data from the filesystem buffer cache to disk, when it awakens at regular intervals.
2. More memory sacrificed to the UNIX buffer cache.

Yet another disadvantage is the inability to use KAIO on Solaris 2.4. See *Use Asynchronous I/O* for more detail.

On an overall basis however, our experience on the Sun Solaris platform, has revealed better overall performance and scalability using raw devices compared to filesystems, depending on the workload. Therefore, in a heavily utilized database configuration we tend to encourage users to use to raw devices.

Moving from filesystems to raw devices:

1. If there is a need to move from UFS to raw, this can be facilitated by utilizing the UNIX `dd` command instead of painstakingly reloading all the data e.g.

```
dd if = /home/myoldUFSfile of = /dev/rdisk/mynewRAWdevice bs=32k
```

Obviously you should remember to:

- Size the raw device appropriately to prevent space wastage.
 - Rename the datafile.
 - Set the permissions on the raw partition appropriately.
2. Also, you may experience, under certain circumstances, a slight degradation in sequential read performance.
 3. On configurations where memory is scarce, you may need to resize your UNIX buffer cache. See section on *Memory and Paging* further on.

4. In the Solaris environment, we encourage the use symbolic links whenever possible since the name of the raw device can change under certain circumstances.g. when reconfiguring disk peripherals or moving the database to a new hardware configuration. Please consult you Sun documentation for more information.

Please refer to the *Oracle7 for Sun SPARC Solaris 2 Installation and Configuration Guide* for more on the operation issues involved in using raw devices.

Getting the best of both worlds:

1. Fortunately, on Solaris you have the freedom to choose UFS for some datafiles and raw partitions for other datafiles since asynchronous I/O works on both raw partitions and UFS. If the nature of I/O activity on database objects is known beforehand, then one could potentially figure out a scheme of placing the datafiles corresponding to specific objects either on UFS or raw partitions, together with a LVM if available.

See *Using readv()* for other side-effects resulting from a mixed (datafiles on UFS and raw partitions) configuration.

The performance gains you can expect using raw devices is around 0 -50%. Under some circumstances (outlined above) for datafiles created on UFS you may experience performance gains from 0 - 40%. Please note that your mileage may vary greatly depending on the nature of your workload and disk or filesystem configuration.

Tip No. 8:Using Direct I/O

Unfortunately, direct I/O is not currently implemented in Solaris 2.4. This may be implemented in the future and will go further in improving the I/O throughput with UFS based datafiles.

III. CPU Scheduling and Process Priorities

Tip No. 9:Use Processor Affinity/Binding on Multi-processor Systems

Binding certain processes to a processor could increase performance substantially. Processor binding is available and fully functional with Solaris 2.

NOTE: In the past, however, with Solaris 2.2, there were some problems with processor binding inheritance where the child processes of a bound process were not automatically bound. This has been fixed.

Processor binding on Solaris 2 is not done automatically. On a multi-processor machine, you need to explicitly bind a process to a processor by using the pbind command (see man pages). Only the super-user or the Oracle owner can bind an Oracle7 process to a processor. Since, pbind is inherited by the child processes this leads to the following considerations:

WARNING! If you are using UFS (with Solaris 2) or raw devices (with Solaris 2.3 and earlier), together with asynchronous I/O, you should never bind the database writer process (dbwr) since the asynchronous I/O employs threads to parallelize the write requests. Therefore, binding the dbwr process to a certain processor will cause all the writes to be serviceable only by one processor! This will lead to a degradation in performance. This can be somewhat alleviated with the ability to bind a process to a group of processors (processor group affinity), a feature that may be available in future releases of Solaris.

However, if you are using raw devices (with Solaris 2.4 and later) together with asynchronous I/O, there is no problem with binding the database writer process. This is due to asynchronous I/O for raw devices being kernelized (KAIO) on Solaris 2.4 onwards, resulting in the parallel asynchronous write operations being done in the Solaris kernel.

In the absence of processor affinity, we recommend binding the various Oracle background processes (except dbwr) to different processors and leaving one processor free to service dbwr. This guarantees the dbwr a processor on which to execute and at the same time allows dbwr to float freely to the other processors as well if it becomes CPU bound.

NOTE: Processor binding is a very complicated issue and should be dealt with care. Processes bound to a processor cannot migrate to different processors even if the latter are free. This might adversely affect the application performance. Because of this, an environment of homogenous applications with balanced load is more suitable for processor binding.

The binding of a process to a processor is not exclusive, i.e. under certain circumstances the processor is free to execute other processes as well.

The general rules of processor binding are to provide higher priority applications with a relatively bigger share of CPU times, to maintain the process context for a longer period, and to evenly distribute the application load across the available processors.

Processor Binding in a Networked Client and Server Environment

When an Oracle client process connects to an Oracle server process through a SQL*Net listener (i.e. running networked client and server Oracle), the server process can be easily bound to a processor by binding the SQL*Net listener process. All Oracle server processes the listener subsequently spawns will automatically be bound to the same processor.

For instance, let's say you have 100 SQL*Forms applications on various client machines communicating via SQL*Net V1 to an Oracle7 instance on a 4-processor server. Of the 100 applications, say applications 1-10 have higher priority. A binding scheme would be to bind the Oracle server processes for the 10 high priority applications to processor 0, and distribute the servers for the rest of the applications evenly across processors 1, 2, and 3. To do so, you would first start 4 orasrv processes (one for each processor) on 4 different ports:

```
% tcpctl port 1256 start &
```

```
[1] 9807
```

```
% tcpctl port 1257 start &
```

```
[1] 9812
```

```
% tcpctl port 1258 start &
```

```
[1] 9833
```

```
% tcpctl port 1259 start &
```

```
[1] 9850
```

Next you would bind each of the listeners to the 4 processors respectively:

```
% pbind -b 0 9807
```

```
% pbind -b 1 9812
```

```
% pbind -b 2 9833
```

```
% pbind -b 3 9850
```

Applications 1-10 will then connect to the database through port 1256, applications 11-40 through port 1257 and so on. For example, for applications 41-70:

```
% setenv TWO_TASK T:dragon/1258:oracle
```

```
% runform
```

Since the orasrv process listening on port 1258 is bound to processor 2, all Oracle servers for applications 41-70 will be bound to processor 2 automatically.

A similar strategy can be adopted for SQL*Net V2.

The performance benefit that is possible is 0-15%. Scalability is enhanced between 20-30% in heavily loaded systems with large number of CPUs.

Processor Binding in a Local Environment

It is more difficult, and probably unreasonable, to use processor binding when the clients and the Oracle servers run on the same machine using the pipe driver two-task. To do so would entail finding the process ID for each server process and manually binding it to a processor. The administrative work would be too excessive to worth the effort, unless the servers have long process lives.

Processor binding of Oracle processes might have negative effect on the performance of non-Oracle applications running on the same machine. Careful tuning is strongly recommended.

Depending on your configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is 0-15%.

Tip No. 10: Tune Spin Count on Multi-processor Systems to Reduce Latch Contention

Making a process goes to sleep after making an initial attempt to acquire a resource turns out to be a costly operation. Alternatively, a process could spin after making the initial attempt, and then go to sleep after the second attempt. This strategy results could yield better performance but could incur more CPU overhead.

The default value of **spin_count** init.ora parameter for Oracle7 for Solaris 2 is 2000 which is adequate for most applications. Increasing spin_count usually results in greater CPU utilization. Setting spin_count to zero (no spinning) is more appropriate for single-processor machines or machines where the CPU utilization is high. We do not usually recommend changing this on Solaris.

Tip No. 11: Real-Time Scheduling

In addition to the normal UNIX scheduler on Solaris, real-time scheduler is also provided for real-time applications in general. Under certain circumstances, e.g. OLTP workloads involving short running transactions, bumping up the priority of a set of processes to real-time may deliver a performance boost.

WARNING! We do not recommend using real-time priority for the Oracle RDBMS or applications in a production environment since the performance can be severely degraded if done haphazardly or incorrectly. Only in very limited situations should this option be considered:

1. A real-time process control applications where this feature may be essential for the proper functioning of the applications.

2. In some environments where the durations of the transactions are well-known and configurable.

The Solaris 2 process scheduler is priority based. A process with higher priority will be scheduled and run before any lower priority processes. Solaris 2 supports three priority classes: timesharing, system, and real-time. By default, all real-time processes have higher priorities than any system process, and all system processes have higher priorities than any timesharing process.

The table below, taken from Sun documentation, illustrates the scheduling policy and priorities. By default, all user (application) processes are in the timesharing

Scheduling Order	Global Priority	Scheduler Class
First	159 . 100	Real-Time
	99 . 60	System
Last	59 . 0	Timesharing

class and system processes (page daemon, etc.) are in the system class. Real-time class is introduced so that critical processes can always get the CPU as soon as it can be run, even if processes in other classes are ready to run or, to some extent, already running. Oracle performance can be improved by putting the Oracle processes in the real-time class.

If you run Oracle server and applications on the same machine and you want to use the real-time class, in general you should put *all* Oracle processes in the real-time class to avoid unbalanced response time. For instance, if you only put the

Oracle server and background processes in the real-time class, and let the SQL*Forms applications stay in the timesharing class, as long as the server and background processes can run, no SQL*Forms application runs. Thus the SQL*Forms users might experience poor response time. You can, however, put the Oracle server, the SQL*Net listener and background processes in the real-time class on one machine and the client processes in the timesharing class on the other machines.

IMPORTANT NOTE: As long as there is a real-time process ready to run, no system process or timesharing process runs. Other real-time processes can run only if they have a higher priority. Real-time processes managed carelessly can have a dramatic negative effect on the performance of non-real-time processes and might result system panic. Putting Oracle processes in the real-time class can seriously affect the performance of other processes that are not in real-time class, including the system processes. Also, real-time scheduling should only be considered, if the Oracle database files are built on raw devices otherwise the pagedaemon, which is in the system class (whose priority cannot be changed), will have a large amount of work to do (since more pages in the UNIX buffer cache could require flushing), thus negatively affecting real-time operation.

The user command `priocntl` (see `man page`) can be used to move a process between timesharing and real-time classes. In order to change the class of a process to real-time, the user invoking `priocntl` must have super-user privilege. Priority classes are inheritable. Another words, child processes by default belong to the same priority classes as their parent processes do.

You can use the `ps` command with the `-c` option to find out what priority class a process belongs to. The class information is under `CLS` column of the `ps` output.

Examples:

The following example shows how you can put Oracle processes in the real-time class.

1) Find out the process ID of the current shell:

```
% echo $$  
2427
```

2) Become super-user and put the shell in the real-time class:

```
# priocntl -s -c RT -i pid 2427
```

3) Start up Oracle instance and SQL*Net listener processes.

Since the shell is in the real-time class, all Oracle background processes and the listener processes will be in the real-time class as well due to class inheritance.

Furthermore, all Oracle server processes spawned by the listener will be in the real-time class as well.

It is more difficult to put application processes in the real-time class since usually they do not collectively belong to a single parent process. Here are some of the ways you can put processes in the real-time class (see man page for `priocntl` for more options):

a) To put SQL*Plus processes with process IDs 3482 and 3483 in real-time class:

```
# priocntl -s -c RT -i pid 3482 3483
```

b) To put the Oracle processes owned by user 'orausr' with user ID 8888 in the real-time class:

```
# priocntl -s -c RT -i uid 8888
```

Note this command will put all processes owned by user 'orausr' in the real-time class, not just the Oracle processes.

c) To put the Oracle processes owned by users in group 'oragrp' with group ID 435 in the real-time class:

```
# priocntl -s -c RT -i gid 435
```

Note this command will put all processes owned by users in group 'oragrp' in the real-time class, not just the Oracle processes.

d) To start a SQL*Plus process in the real-time class:

```
# priocntl -c RT -e sqlplus
```

Adjusting Priority within Real-Time Class

You can also adjust priorities within the real-time class. You can use this feature to give certain Oracle processes higher priorities than others within the real-time class. Again, you should fully understand the effects before proceeding.

We have experienced 10-15% gains in performance for some OLTP workloads.

IV. Memory and Paging

Tip No. 12: Controlling Buffer-Cache Paging Activity

Excessive paging activity hurts performance significantly. This becomes a more significant issue with databases created on UFS files. In this situation, a large

number of SGA data buffers may also have analogous UFS buffers containing the most frequently referenced data. The behavior of the daemon that flushes filesystem buffer caches to disk can have a significant impact on performance due to the frequency with which it is woken and the amount of work that it has to accomplish. This could pose to be an I/O bottleneck resulting in lower overall system throughput.

In Solaris 2, the kernel parameters `tune_t_fsflushr` and `autoup` control the behavior of this daemon, called `fsflushr`. At regular intervals, `fsflushr` will go through a fraction of the page structures every `tune_t_fsflushr` and cycle through all the page structures in `autoup` seconds. For example, if `tune_t_fsflushr` is set to 60 and `autoup` to 180, then the `fsflushr` will look at 1/3 of the buffer cache every time it wakes up, every 60 seconds. The whole buffer cache will be flushed in 180 seconds. The values you pick will depend on how much you wish to alleviate the paging activity on your system.

Depending on the nature of the workload this could yield a performance improvement of 0 to 15%.

Tip No. 13:Configuring Swap Space

Inadequate swap space usually results in the system hanging or having abnormally slow response times. On Solaris 2, swap space can be dynamically configured on raw partitions and/or filesystem files. The amount of swap space to configure depends on the amount of physical memory present and the swap space requirements of your applications.

The general rule of thumb is to configure this to be 2 to 3 times the amount of physical memory for almost all workloads.

Tip No. 14:Setting the Database Blocksize

The Oracle database blocksize can be configured for better I/O throughput. In the Solaris 2, the logical blocksizes of the filesystems depend on the way the filesystem has been generated. This is usually 2K, 4K or 8K. If the Oracle database is UFS based then the block size should be some multiple of this. On Solaris, the `db_block_size` ranges from 2K (default) to 8K. (current maximum). Similarly for databases on raw partitions the Oracle database block size should be some multiple of the operating system physical block size (512 bytes on Solaris).

We recommend a smaller Oracle database blocksizes (2K or 4K) for OLTP or mixed workload environments, and larger blocksizes (8K) for DSS type workloads.

You may experience overall performance improvements ranging from 0 to 5%.

Tip No. 15: Reduce Redo Log Buffer Latch Contention

In order to reduce contention for redo log buffer latches, one can reduce the time the latch is held and in a multiprocessor environment, simply add more latches. The former can be achieved by reducing the value of the init.ora parameter **log_small_entry_max_size** and the latter by changing the value of the init.ora parameter **log_simultaneous_copies**. By default **log_simultaneous_copies** is set to the number of CPUs a machine has available. We recommend setting this to anywhere between its default value to two times of the default value i.e. twice as the number of CPUs available and reducing the **log_small_entry_max_size** to zero.

The performance gains depend on amount of redo information generated by transactions and your system utilization.

Tip No. 16: Tuning the Archiver Buffers

Dedicating more buffers and larger buffers for archiving changes to the database. Overconfiguring these values could result in a faster archiver but at the expense of degraded overall system performance

The init.ora parameters **log_archive_buffer_size** (default 4) can be bumped up to a maximum of 128. The default for **log_archive_buffers** is 4.

Tune this carefully, so that overall performance of normal database activity is not degraded drastically. This may result in performance improvements in the archiving process from 0 to 20%. Some users have reported larger improvements.

Tip No. 17: Using Intimate Shared Memory (ISM)

Sun's Intimate Shared Memory feature allows different processes attaching to the same shared memory address to share the same page table. This avoids page table stealing and thrashing and thus enhances database performance. ISM is supported for Sun-4m and Sun-4d systems only. Use the Solaris command **arch -k** to determine the architecture of the machine you are running on.

Currently on Solaris 2, the ISM feature is incorporated in the operating system kernel and is enabled by default on Solaris 2.3 onwards. Applications still need to explicitly use this feature to take advantage of it. Oracle7 on Solaris takes full advantage of this feature by default (i.e. the init.ora parameter **use_ism=TRUE** by default). Setting **use_ism=FALSE** turns off this facility but may degrade performance. Turning on this feature on architectures that do not support this facility does not have any effect.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 - 200%

Tip No. 18: Tuning the SGA Size

The maximum size of a shared memory segment SHMMAX is constrained by the datatype that is used to represent this parameter, which is an integer in our case resulting in a maximum size of 2 Gigabytes for a single shared memory segment. The other constraint is the amount of virtual memory addressable by the architecture.

Therefore the constraining factor in using multiple shared memory segments for the SGA becomes amount of physical memory that current hardware architectures can support. Obviously, we would not like to see the Oracle SGA paged or swapped out because of inadequate main memory. For a large number of simultaneous database users, larger SGAs are recommended. Very large SGAs (in the order of gigabytes) may require relocation of the SGA base address. Please refer to the *Oracle7 for Sun SPARC Solaris 2 Installation and Configuration Guide* for more information on configuring the SGA.

The init.ora parameters that have the greatest effect on the SGA size are **db_block_buffers** and **shared_pool_size**. Bumping up these parameters depends on the amount of memory you can sacrifice to the SGA.

Tip No. 19: Tuning the UNIX Buffer Cache

The Solaris filesystem buffer cache plays an important part in performance when UFS based datafiles are used. Since the purpose of the UNIX buffer cache is to reduce disk access frequency, if this cache is too small, then disk utilization will increase and potentially saturate one or more disks. (See also the tip for *Controlling Buffer-Cache Paging Activity* above, for the implications of increasing this parameter.) On the other hand, if this is too large then precious memory is being wasted.

The UNIX buffer cache on Solaris 2 can be configured via the kernel parameter **bufhwm**. Bufhwm is the maximum amount of physical memory, in kilobytes, that can be used by the I/O buffers; it limits the number of buffers that can exist at any time. By default, up to 2% of the system memory is used, this can be increased to 20%. In general, if the buffer hit ratio is low (less than 90%, see sar -b), increasing bufhwm could help. If maintaining a high buffer hit ratio is not critical, decreasing bufhwm will make more physical memory available. Refer to your Solaris documentation for more information on this topic.

The performance gain cannot be quantified easily since it depends on the degree of multiprogramming and I/O characteristics of the workload.

Tip No. 20: I/O Buffers and SQL*Loader

For high speed data loading (e.g. using SQL*Loader direct path option in addition to loading data in parallel), the CPU spends most of its time waiting for I/O operations to complete. By increasing the number of buffers one can usually push the CPU utilization harder thereby increasing overall throughput.

The number of buffers (SQL*Loader parameter BUFFERS) will depend on the amount of memory you have available and how hard you wish to push CPU utilization. Please refer to *Oracle7 Server Utilities User Guide V7.1* for information on adjusting the file processing options string for the BUFFERS parameter.

Your performance gains will depend on your CPU utilization and the degree of parallelism that you use when loading. Please refer to the appropriate chapters in the *Oracle7 Server Documentation Addendum REL 7.1* for more details on parallelising the load operation and the *Oracle7 Server Utilities User Guide V7.1* for more generic information on the SQL*Loader utility.

V. Network / IPC tuning**Tip No. 21: Use out-of-band breaks with SQL*Net TCP/IP**

On Solaris 2, the TCP/IP networking software supports the mechanism utilized by SQL*Net TCP/IP for out-band breaks. With SQL*Net V1 the orasrv program is set up to negotiate out-band breaks by default. This is also true for the SQL*Net V2 listener process.

Depending on how often SQL*Net communication is taking place between a client and server, the improvement in performance with using out-band breaks is 0 - 20% over using in-band breaks. Some users have seen even bigger improvements.

Tip No. 22: Use the Post-Wait Driver

The post-wait driver reduces the overhead incurred by the more expensive use of semaphore operation for interprocess communication. The post-wait driver is not yet implemented for ORACLE on Solaris although efforts are currently underway towards implementing this feature. At this time we cannot fully quantify the performance improvement, other than to say that it will reduce the overhead incurred by high interprocess communication activity common on OLTP databases. More extensive experiments need to be completed to determine the performance gains more precisely under different workloads.

Tip No. 23:TCP/IP Performance Issues

Solaris 2.4 has sped up network calls considerably, compared to Solaris 2.3. These are ultimately realized as performance gains and improved scalability of workloads that incur high network traffic.

The packet size utilized internally by SQL*Net V2 on Solaris 2 is 2K by default. The packetsize utilized by the underlying network layer (TCP/IP) is 1K for most installations. This packet size is usually equal to or is some multiple of the underlying TCP/IP packet size/s, otherwise this may generate unnecessary network overhead. This SQL*Net packet size or buffer size can be manipulated by specifying a size in bytes in the SQL*Net connect string. However, we do not recommend that users override this SQL*Net buffer size explicitly because:

1. There have been no hard cases where this clearly provides a significant boost in network performance.
2. Sometimes, this may even result in significant performance degradation.

VI. Additional Tuning Tips

This section provides additional performance tuning tips that may not be covered in the *Oracle for UNIX Performance Tuning Tips*. The tips are mostly Sun specific.

Tip No. 24:Compiling Pro*C Programs

Using SC3.0.1 C compiler from Sunsoft, some compiler flags may improve the performance of Oracle applications written using Pro*C or Oracle OCI interface e.g.

- Try to use the highest level of optimization (-xO4).
- Try to use the “-fast” compiler directive.

Depending to on the nature of the application, these more aggressive optimizations may result in better performance. Please refer to your compiler documentation for more information on these flags.

Tip No. 25:Some Useful Performance Monitoring and Tuning Tool/Utilities

There is a plethora of monitoring tools available on Solaris:

- SPARCworks from Sunsoft for performance tuning 3GL applications

-
- Proctool 2.3/4 is available as an unsupported product from Sun for Solaris 2.3/4 respectively, for performance monitoring. Extremely quick to install and use.
 - Opentune 2.0.1 from Amdahl for Solaris 2.3/4 for performance monitoring.
 - Other GUI based performance tools like Glance from HP etc.

Tip No. 26: Useful Configuration Files on Solaris

Solaris 2 kernel

/etc/system

LVM: Online Disksuite 2/3/4.x

md.tab

LVM: Veritas Volume Manager

use the vxprint utility

VII. Useful References

1. Oracle for UNIX Tuning Tips (Part# A22535)
2. Oracle7 for Sun SPARC Solaris 2 Installation and Configuration Guide (Part# A14783)
3. Oracle7 Server Administrators Guide (Part# 6694-70-1292)
4. Oracle7 Server Concepts Manual (Part# 6693-70-1292)
5. Oracle7 Server SQL Language Reference Manual (Part# 778-70-1292)
6. Oracle7 Server Documentation Addendum Rel 7.1 (Part# A12042)
7. Oracle7 Server Utilities User Guide Rel 7.1 (Part# A12389)
8. Oracle7 Server Application Developer's Guide (Part# 6695-70-1292)
9. Solaris 2.3 Answerbook
10. SPARCCompiler C 3.0 Answerbook
11. SPARCworks/SPARCcompiler 3.0 Answerbook
12. Sun Performance Tuning Overview (Part# 801-4872-07) October 1993
13. Sun Performance and Tuning: SPARC and Solaris (ISBN 0-13-149642-3)

VIII. Feedback

Include the following when providing feedback on the quality and usefulness of this document:

- ATTENTION: SUNINFO
- SUBJECT: Oracle for Sun Performance Tuning Tips
- errors encountered
- clarity and readability
- where more technical depth can be provided
- quantity and quality of examples provided
- other features or areas that need to be covered

EMAIL: suninfo@oracle.com

FAX: (415) 506-7223 (Attention: SUNINFO)

MAIL: Suninfo

Sun Products Division (Performance Group)

Oracle Corporation

500 Oracle Parkway

Box 659407

Redwood Shores

CA 94065, USA

Part #: A25584-1

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Worldwide Inquiries
415.506.7000
Fax 415.506.7200