



Avoiding Common Performance Issues When Scaling RDBMS Applications With Oracle 9i Release 2 and Sun Fire™ Servers

Glenn P. Fawcett, Strategic Applications Engineering

Sun BluePrints™ OnLine—March 2003



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

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

Part No. 817-1781-10
Revision 1.0, mm/dd/02
Edition: March 2003

Copyright 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, 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 document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

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

Sun, Sun Microsystems, the Sun logo, AnswerBook2, docs.sun.com, BluePrints, Sun Fire, Sun Quad FastEthernet, UltraSPARC, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and in other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and in 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 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, 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 déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, AnswerBook2, docs.sun.com, BluePrints, Sun Fire, Sun Quad FastEthernet, UltraSPARC, et Solaris sont des marques de fabrique ou des marques déposées 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 licenciées de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.



Please
Recycle



Adobe PostScript

Avoiding Common Performance Issues When Scaling RDBMS Applications With Oracle 9i Release 2 and Sun Fire Servers

There are a handful of common performance issues that crop up when trying to scale Oracle database applications on Solaris Operating Environment (OE). These problems are sometimes tough to identify and address. This paper incorporates the experiences of the Strategic Applications Engineering (SAE) group in tuning Oracle RDBMS systems in a variety of situations.

The purpose in documenting these common themes in Oracle performance, is so that the the reader could recognize and ultimately avoid these situations. These issues are valid in Oracle 6.0 through Oracle 9i Release 2; any differences among Oracle versions are noted.

This paper details:

- Solaris and Oracle tools required to monitor for these situations. Example output and sample reports are included in the Appendices.
- Oracle 9i Release 2 availability features and Real Application Custers (RAC) are discussed.
- Common performance issues. These issues range from hard to identify and rectify to well understood and easy to fix. Diagnosis techniques and fixes are explored.

The issues explored in this paper are as follows:

- Shared pool and library cache latch contention
- Cache buffers chains latch contention
- Schema issues for insert and update intensive applications – Special consideration must be taken to avoid contention when scaling these applications.
- Network Issues – The interplay of hardware, operating system, and Oracle SQL*net settings for optimal performance

- Checkpoint tuning – Checkpoints are critical to ensure availability. On large systems, checkpoints can become a performance burden if not configured properly.
- Data layout issues – Striping schemes and redo-log placement are discussed.

Appendices to this article can be found in the accompanying BluePrints document “Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices”.

Overview of Performance Monitoring Tools and Techniques

This section describes tools and techniques used to gather Oracle and the Solaris operating environment (Solaris OE) performance statistics. It is by no means an all-inclusive list of performance tools. Readers are encouraged to use whatever is best for their environments. The tools described here can be used to diagnose problems presented by examples.

Oracle Statistics

Oracle statistics are gathered from the V\$ tables. These tables hold statistics on latches, wait events, IO, and numerous other statistics useful in identifying Oracle performance issues. There are two standard Oracle methods that summarize performance information using the V\$ tables. Regardless of the method, the first thing to look for is *wait events*.

Wait events are counted and summarized for each Oracle server process. Wait events give tremendous insight into what is causing a system bottleneck. For example, processes are expected to wait for IO to complete, but if processes are waiting longer to obtain a latch, there is a problem.

`utlbstat.sql` and `utlestat.sql`

Prior to Oracle 9i, Oracle's standard method of extracting performance information from a running RDBMS system uses two scripts in the `$ORACLE_HOME/rdbms/admin` directory: `utlbstat.sql` and `utlestat.sql`.

The `utlbstat.sql` script is run at the beginning of the measurement interval. The `utlbstat.sql` script creates temporary tables, and stores results of V\$ table queries in these tables. After the measurement interval is complete, the `utlestat.sql` script is run. `utlestat.sql` queries the V\$ tables, compares the results with the stored results from the `utlbstat.sql` script, and removes temporary tables. Results are formatted by `utlestat.sql` and stored in the `report.txt` file. A sample is included in Appendix A of the accompanying BluePrints article “Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices”.

Performance data in `report.txt` is averaged over the entire interval. This basic method gathers good general information but fails to do a good job at summarizing workload rates. The best part about this report is that it is on every Oracle version since 6.0 and nearly every DBA knows how to run it.

Oracle StatsPack

This package was officially introduced in Oracle 8i (8.1.5). Like the `utlbstat.sql` and `utlestat.sql` scripts, StatsPack uses the V\$ tables to gather performance data. StatsPack is series of packages and reports that take *snapshots* of the V\$ tables and store them in database tables for future analysis. The report script `ORACLE_HOME/rdbms/admin/spreport` is used to generate a report by taking the difference between any two snapshots.

There are numerous advantages over `bstat/estat` to using this new statistics gathering method:

- Excellent workload profiling with per second and per transaction rates
- Top 5 list of wait events
- Persistent storage of performance data in database tables
- Less performance impact when gathering a *snapshot*
- Ability to export data and store offline
- Details on parent/child latches

A sample StatsPack report is included in Appendix B of the accompanying BluePrints article “Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices”. For information on how to install and use the StatsPack package, refer to the readme file: `ORACLE_HOME/rdbms/admin/spdoc.txt`.

Additionally, there are references to Oracle articles on StatsPack in “References” on page 25.

Solaris OE Performance Monitoring

There are a variety of tools that can be used to monitor Solaris OE. Many papers, presentations, and books have been written on Solaris OE performance tuning and monitoring. The following list describes a basic set of tools as a guideline.

- `netstat(1M)` is used to monitor individual network interfaces. This data is useful in determining whether a network interface is overloaded or not.
- `iostat(1M)` is used to collect disk drive statistics such as response time and queue length.
- `vxstat(1M)` is a Veritas utility that displays IO statistics at the volume level.
- `vmstat(1M)` can be used to gather statistics on run queues, swap usage, paging, context switches, and CPU utilization.
- `sar(1M)`, system activity reporter, can gather most OS statistics from CPU usage to disk response times.
- `mpstat(1M)`, multiprocessor statistics, provides detailed statistics on processors that include: voluntary/involuntary context switches, syscalls, interrupts, cross calls, and CPU utilization.
- `prstat(1M)` is like the popular freeware `top` command. Processes using the most CPU are shown ahead of other processes.

Common Performance Issues

The following common performance issues have been seen in numerous customer and benchmark situations. Documenting these issues will help you to better understand and, hopefully, avoid them.

- Shared pool and library cache latch contention.
- Cache buffers chains latch contention.
- Schema issues for insert and update intensive applications. Special consideration must be taken to avoid contention when scaling applications.
- Network issues describes the interplay of hardware, operating system, and Oracle SQL*net settings for optimal performance.
- Checkpoint tuning is critical to ensure availability. On large systems, checkpoints can become a performance burden if not configured properly.
- Data layout issues – Striping schemes and redo log placement are important.

Shared Pool Contention

The shared pool was introduced in Oracle 7.0 to allow user sessions to share SQL statements. Before an SQL statement is run, the shared pool is checked. If a matching SQL statement is found, CPU cycles can be spared by avoiding a parse. Statements that use bind variables and procedures tend to stay in the cache and be reused. SQL statements that set explicit values typically do not match and are aged out of the cache.

This scheme works quite well, giving most applications a substantial benefit from caching SQL statements. However, there are some applications that show little or no benefit. These applications construct a fully qualified SQL statement and submit it to the database for parsing and execution. These applications tend not to reuse SQL statements.

The problems were not that great when Oracle 7.0 was introduced, but by the time servers had increased throughput by nearly 100 percent and the base applications had not changed, there were problems. Soon, it was taking longer to search for matches and flush an unused statement than it was to *reparse* the statement in the first place.

DBAs noticed that on a fresh restart of the database, the shared pool latch did not experience many sleeps, and users had consistently good response times. As the number of SQL statements in shared pool grew, latch sleeps got out of hand and response times grew exponentially.

It was common for People Soft and Oracle Financials DBAs to issue the SQL command `alter system flush shared pool;` on a regular basis. This would flush all SQL statements from the shared pool, thereby shortening subsequent searches.

Initially, there was confusion about how to fix this situation. Some would make the shared pool bigger. If there was little reuse, this fix would make the problem worse by increasing the number of statements to search.

Recognizing Shared Pool Contention

As more users access the system, response times rise as contention on the shared pool latch increases. CPU usage is sporadic as Oracle processes rush to obtain the shared pool latch only to spin and go to sleep.

Oracle's statistics clearly show shared pool contention in the following ways:

- High sleeps on the shared pool latch and library cache. Both the `report.txt` and StatsPack reports show this situation.
- High number of parses per transaction rate. Beginning with Oracle version 8.0, parses are broken out by hard and soft. The hard parses are the ones to watch.

- StatsPack reports show hard and soft parses in the `Load` profile section. Parses are identified under the `instance efficiency` section. A full report on how much memory is allocated by each component of shared pool is shown.
- Issue the SQL command `alter system flush shared pool;`. If the system is more responsive after this command completes, chances are the database is experiencing shared pool contention. Appendix A shows a `report.txt` file with shared pool contention. Appendix A can be found in the accompanying BluePrints article “Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices”.

Reducing Shared Pool Contention

As with most performance problems, there are multiple ways to address shared pool contention. Absolutely the best way to fix it is to change the application. Unfortunately, changing the application is often not trivial.

Developers and vendors must change their queries to use bind variables or packages. Sometimes, the code is not in-house or it is a third-party application. The reader should be lobbying for application re-coding, but in the meantime, here are a few things that can be done.

`spin_count`

Increasing the `spin_count` `init.ora` parameter can help. A detailed discussion can be found in the general latching section, “`init.ora: spin_count`” on page 25.

`cursor_sharing`

In Oracle 8.1.6, the concept of automatic cursor sharing was introduced. This feature transforms SQL statements by replacing static values with dummy bind variables. This feature can be set dynamically or in the `init.ora` file.

With cursor sharing enabled the SQL statement:

```
select name from emp
where empid=875;
```

will be transformed to:

```
select name from emp
where empid=:sysb1
```

Cursor sharing can be set dynamically in a `svrmgr` session:

```
alter system set CURSOR_SHARING=FORCE;
```

Note – You may also have to set the hidden `init.ora` parameter `_SQLExec_PROGRESSION_COST=0` on Oracle versions (8.1.6 - 8.1.7). This parameter controls the cost monitoring of SQL statements eligible for sharing. Setting `_SQLExec_PROGRESSION_COST=0` will disable this feature and cause all statements to be shared. (Metalink docid: 68955.1)

Shared Pool Flushing

Shared pool flushing can reduce the time to acquire a latch. Techniques deployed at numerous Oracle sites range from writing daemons to flush hourly or, based on load, to manual intervention when user response times become unacceptable.

Flushing the shared pool is pretty harsh since it causes all activity to halt until it is complete. Good SQL statements and system packages will be flushed along with the unusable ones. For a few minutes, there will be more parsing than usual. There are several clever scripts from Steve Adams <http://www.ixora.com.au/> that pin SQL statements from system and user packages in the shared pool, then perform a flush. This *nice flush* is less obtrusive.

`session_cached_cursors`

If the application is doing a lot of recursive parsing of cursors, this parameter will cache cursors on a per session basis and therefore prevent subsequent parses and accesses to the shared pool. This sort of contention also shows pressure on the library cache latch.

By default `session_cached_cursors` is set to 10. For forms-based applications, this is too low. For Oracle financial and manufacturing suites use:

```
session_cached_cursors = 200
```

Downsize the Shared Pool

If reuse is poor, it doesn't make sense to have a huge shared pool unless you are flushing the pool on a regular basis, before it becomes full. Often shrinking the shared pool can yield substantial performance improvements.

The `V$sgastat` table can be used to dynamically monitor growth of the shared pool. If shared pool growth is suspected to be a problem, scripts that poll this table on a regular basis can be useful for identifying unanticipated growth.

In Oracle 8i, the parameter `db_block_hash_latches` was introduced. This parameter is used to protect a number of hash chains. Generally, the default of 8i is best. There are fewer latches, but more chains and thus shorter chain walks. In large systems, however, this may not be enough.

By default there are 1024 latches per instance. Prior to Oracle 8i, there was a one-to-one relationship of hash chain latches to `_db_block_hash_buckets`.

Recognizing Cache Buffers Chains Latch Contention

Looking at OS statistics is not conclusive for identifying cache buffers chains latch contention. Elevated levels of context switches are symptomatic of this problem, but this alone is not conclusive. The easiest way to determine if "cache buffers chains" is a problem is with Oracle statistics.

Whether using `report.txt`, StatsPack, or third party monitoring tools, this should be easy to spot. There is an example of a StatsPack report that shows this performance problem in Appendix B in the accompanying BluePrints article "Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices". This problem shows up in Oracle statistics as the following:

- The latch free wait event will be high.
- High cache buffers chains latch sleeps.

Identifying cache buffers chains latch contention is easy, but the solution requires further investigation.

There are multiple solutions depending on which sort of contention is taking place. Three common situations that can lead to contention on the cache buffers chains latch are:

- same chain contention – Contention for different blocks hashed to the same chain
- same block contention – Contention for the same database block
- same row contention – Contention for the same database rowid

Start investigation by examining which objects are on chains with high sleep counts.

Figure 3 shows a SQL script that finds cache buffers chain latches with a high number of sleeps. For this example, the number of sleeps is set to 1000, which should be adjusted as needed. This script identifies objects that are on the most

contentious chains. Either the chain length is the problem, or the objects under them are. This example shows an over-use of the DUAL and MTL_ONHAND_QUANTITIES objects. The application might be able to be modified to avoid stress on these objects.

```

column object_name format a24

select HLADDR, DBARFIL, dbablk, tch, object_name
from X$BH b, all_objects o
where HLADDR in
(
select ADDR from v$latch_children
where sleeps > 1000 and
name like '%cache buffers%'
) and
b.obj = o.object_id
order by HLADDR
/

SAMPLE OUTPUT:::

HLADDR      DBARFIL      DBABLK      TCH OBJECT_NAME
-----
66194464         1         417         119 DUAL
66194464        16       131999          0 SO_LINES_ALL
66194464        16       131999          1 SO_LINES_ALL
661FDC98         4       133306         34 MTL_ITEM_CATEGOR
                    IES_U1
661FDC98        16       133714         48 MTL_ONHAND_QUANT
                    ITIES

```

FIGURE 2 Finds Chains With High Sleep Count Script and Output

If there are multiple DBABLK block numbers that are the same, then contention is at the block or row level. Row level contention will also exhibit high enqueue waits.

The SQL script in Figure 4 can be used to find statements which are blocked from running due to an enqueue.

Finally, if chains are long and DBABLK are different, then same chain contention is occurring.

Resolving Same Chain Contention

The `init.ora` parameter `_db_block_hash_buckets` controls the number of hash buckets. By default, the number of buckets can be too small for large systems. Also, in Oracle 8i, the structure was changed to have multiple chains protected by a single latch. To see if the technique will resolve the contention, use the following `init.ora` parameters as a starting point:

Oracle 8.1.5 and greater

```
_db_block_hash_buckets = Prime(4 X db_block_buffers)
```

where Prime is a prime number about the size specified

```
db_block_hash_latches = 8192
```

The default is 1024.

Before Oracle 8.1.5

```
_db_block_hash_buckets = Prime(4 X db_block_buffers)
```

where Prime is a prime number about the size specified

Increasing the `spin_count` can also help. See section on `spin_count` “`init.ora: spin_count`” on page 25.

These changes have little risk. The chains are made shorter and more latches are created. This will help reduce the time hash latches are held and, therefore, latch contention will be reduced.

If this doesn't reduce `cache buffers chains latch sleeps`, then there is most likely contention for the same block or row.

Resolving Same Block Contention

Oracle Statistics can show block contention pretty easily. Both `report.txt` and `StatPack` will show a high number of buffer busy waits if you are seeing same block contention. Figure 3 shows blocks with a high sleep count.

Contention for the same block can be resolved by applying the techniques used for heavy insert/update applications covered in “Schema Issues for Heavy Insert/Update Applications” on page 12.

Resolving Same Row Contention

This requires changing the application, there is seldom an easy fix.

Sometimes same row contention occurs with applications that update counters or statistics. These situations can often be adjusted using application parameters or simple re-coding using ranges of sequence numbers or the sequence datatype.

Schema Issues for Heavy Insert/Update Applications

There are several performance problems that arise when scaling heavy insert/update applications. These problems are easily recognizable and avoidable.

Recognizing Schema Contention Issues.

Schema contention is identified by Oracle wait events. The wait events section of the Oracle performance statistics report (`report.txt` or StatsPack) will show high `buffer busy` and `enqueue` waits. Some waits on `enqueue` and `buffer busy` are normal in this environment, but they should not be the dominant event.

The `buffer busy` wait events occur when two processes want to access or modify the same buffer. One process will get the buffer and the other will wait for the `busy` buffer. There are several remedies to this situation, but first the origin of the contention must be determined.

The `report.txt` and StatsPack reports show `buffer busy` waits on a file-by-file basis. This level of granularity might be enough if this data file belongs to a tablespace with one object. This may be the case with *large* tables or indexes. If there are multiple objects in this tablespace, further detail will be required. The `report.txt` example in appendix C of the accompanying BluePrints article “Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices” shows `buffer busy` wait issues.

By sampling the `v$session_wait` table for `buffer busy` wait events, it is possible to determine the `file#` and `block#` using the `p1raw` and `p2raw` columns of the `v$session_wait` table. Once the `file#` and `block#` are found, this can be used to determine the object.

Sample scripts that help determine the objects that are getting `buffer busy` waits are included in Appendix D of the accompanying BluePrints article “Avoiding Common Performance Issues When Scaling RDBMS Applications with Oracle 9i Release 2 and Sun Fire™ Servers Appendices”. These scripts sample data and insert results into a database table with time stamp information. From this data, reports are written to show which objects exhibit the most `buffer busy` waits. Additionally, histograms of wait events versus time can be determined.

An alternate quick and dirty way to determine the objects that are in contention, is to find the SQL statements that are being blocked. This is done by querying the `v$sqltext` table. Figure 4 prints out the statements from the shared pool that are blocked due to an enqueue.

```
select sid, sql_text
       from v$sqlsession s, v$sqltext t
where
  s.sql_hash_value=t.hash_value and
  s.sid in
  (select sid
   from v$sqlsession_wait
   where event = 'enqueue')
order by sid,piece
/
```

FIGURE 3 Find Statements Blocked on Enqueues

Resolving Schema Contention Issues

There are a number of techniques that are useful in resolving schema contention issues for high insert/update applications.

Freelists and Freelist Groups

Oracle default storage parameters are fine for 1 to 4 CPU systems, but completely inadequate for systems with 24 to 64 CPUs.

The `init.ora` parameters `freelist` and `freelist groups` are used to help provide concurrent access for processes to insert into the same table. By default, Oracle creates *one* freelist and *one* group per object. This works reasonably well for small CPU counts, but doesn't scale to 64 CPUs.

The `freelists` and `freelist groups` parameters are set in the storage clause when creating tables and indexes. The number of *total* `freelists` is:

`freelists * freelist groups`

```
create table hdata (  
    h_id    number,  
    h_date  date,  
    h_data  varchar2(24)  
) tablespace misc  
  initrans 4  
  pctfree 5  
  storage  
    (freelist groups 43  
     freelists 9  
     buffer_pool keep);
```

FIGURE 4 Important Table Attributes

For insert intensive tables, you should set the `freelist groups` to at least the number of CPUs. The `freelists` parameter should be greater than one but is typically less than the number of CPUs. In Oracle 9i Release 2, you can create locally managed tablespaces and allow Oracle to manage segment allocation automatically. Automatic segment-space management manages freespace in a tablespace as a bitmap and not lists. `pctused`, `freelist`, and `freelist groups` parameters are ignored if these Automatic segment-space management is used.

Initrans

`Initrans` is a table attribute that defines the number of concurrent changes that can be made to a block. If this attribute is too low, it can cause updates to stall on buffer busy waits.

High Watermark

To insert into a table, a process must obtain blocks from a `freelist`. To obtain a `freelist`, a high watermark enqueue must be obtained. In insert intensive applications, the high watermark enqueue can become a bottleneck.

High watermark enqueues must be obtained each time the high watermark is hit. By default, the high watermark count is set to zero. This parameter sets the number of blocks allocated per free list on advancing high watermarks.

For example, setting the `init.ora`:

```
bump_highwater_mark_count = 300
```

will reduce the number of times the high watermark enqueue is obtained by 300 times!

Scripts to identify locks or enqueues sometimes refer to high watermark enqueues as space management enqueues or space management transactions. Oracle Metalink Docid: 1020008.6 contains a great script `tfsclock.sql` that fully decodes locking information.

Locally Managed Tablespaces

Locally managed tablespaces do not require updates to the catalog tables when space is needed and, therefore, a HW mark enqueue is not required. All extent management is handled locally. Locally managed tablespaces are common for large systems and required for exportable or read-only tablespaces.

Figure 6 shows initial creation and migration to locally managed tablespaces.

```
SQL> create tablespace XYZ
      datafile '/d01/xyz1.dbf'
      size 100M
      extent management local
      uniform size 512K
      nologging;

SQL>exec
dbms_space_admin.tablespace_migrate_to_local('XYZ');

SQL> alter tablespace XYZ minimum extent 1M;
```

FIGURE 5 Creating and Altering Existing Tablespaces to be Locally Managed

Sparse Indexes

This technique uses the `PCTFREE` storage parameter to create indexes that initially fill only a small portion of a block.

For example:

```
create index id1 on abc (id) pctfree 80;
```

This leaves 80% free space in the block. This will allow for more inserts to an index before splitting the index into new blocks. This technique can also reduce "cache buffers chains" contention by spreading an index over more blocks. If there are more blocks, the likelihood of contention on a particular chain is reduced. PCTFREE = 80 will leave 80% of the block free to allow for growth. Periodic recreation of the index is required if index splits begin to degrade performance.

For example:

```
alter index id1 rebuild pctfree 80;
```

Alternate Structures for Tables and Indexes

One approach to decreasing schema contention is to change the access method. Alternate schema representations can have a profound effect on the way the data is accessed. Alternate structures that are useful for modifying intensive environments include:

- Index only tables (IOT) – IOTs are basically a primary index where the leaf nodes contain the actual table data. This structure can reduce the amount of manipulation upon update and insert. Of course, if possible, a table with NO indexes is best for insert; this is not too common because all accesses would then require a full table scan.
- Clustered tables – Clustered tables organized as hash work well for heavy update environments where the table doesn't grow much. Clustered tables with hash are limited to single or multipart primary keys that are comprised of integers. Hash tables are not prone to grow in a civilized manner (chaining), so their use should be restricted to tables that do not grow too much. Space can be preallocated if the number of rows can be predicted.
- Partitioned tables – Table partitioning with local indexes allows for distributed index manipulation. Partitioned tables that use global indexes will not reduce contention. For example, consider what happens upon insert to a table with B-tree indexes. The B-tree has to be parsed and modified. Multiple locks must be taken out as processes walk down the tree. With a large single index, it is easy to see how hundreds of users could run into contention on the upper level B-tree nodes. Partitioning effectively removes the top index nodes by applying a range or hash function.

Network Issues

Severe network issues can sometimes be identified using Solaris tools. The `netstat(1M)` utility is very good at eliminating the network as an issue, but it is not so good at isolating actual network issues.

Note – The `netstat(1M)` utility reports packets per interval, not packets per second. For intervals greater than one second, you will need to divide to get packets per second.

There are several places where a degradation in network performance can affect user response times.

Network Interface Controller Contention

It is important to make sure the physical network is as fast as possible. Use a fast network switch to connect multiple clients. To connect an application server to a database server, it is often desirable to use a *cross-over* cable to create a private network between the two machines. Once a network topology has been chosen, it is important to monitor the performance.

The `netstat(1)` utility can be used to calculate number of packets/second as seen by the Network Interface Controller (NIC). The Sun Quad FastEthernet (qfe) and Gigabit (ge) NICs can comfortably deliver up to 15,000 packets/second with UltraSPARC II 400 MHz processors. A controller that is consistently overloaded or shows a high number of collisions is suspect. If the NIC is overloaded, a second controller and Oracle listener can be configured.

The new network interface can be trunked with the first and share the same IP address, or a new IP address can be assigned. If a new IP is assigned, it is easy to configure an Oracle listener to use this new IP address.

Sql*Net Issues

Even if the physical network is performing well, it is possible for Oracle clients and servers to run not as efficiently as possible. This happens when the logical and physical sizes of packets do not match and so packets are split at the NIC layer or SQL*Net layer.

Oracle sets the session data unit (SDU) in Sql*net to 2048 bytes by default. This works fine if the data being delivered is less than 2048 bytes. If the amount of data to be delivered is greater than the SDU, the Oracle client and server will break this into multiple packets. The Oracle server process will incur a wait event that causes it to context switch and wait until more data is available.

Identifying Sql*net Waits

Sql*net wait events are found in `report.txt`, `StatsPack`, or can be gathered for an individual session via the `v$session_wait` table. The `v$session_wait` table is useful when tracking down which sessions are causing the `sql*net` more data from/to `client` wait events. For example, a batch job could be causing these wait events. If possible, the batch clients could be configured on a different listener with a modified SDU.

Maximum Transport Unit, Transport Data Unit, and Session Data Unit Sizes

It is best to start off with some clear definitions of these values and how they relate to Sql*Net performance.

- Maximum transport unit (MTU)

This is the value supported by the OS or network hardware. Solaris supports an MTU of 1500 bytes for Ethernet. Using the `ifconfig(1)` command, you can display the value set on a particular interface (Figure 7).

```
root@repl> ifconfig qfe0
qfe0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500
index 2 inet 129.153.41.170 netmask ffffffff0 broadcast
129.153.41.255 ether 0:0:be:a7:0:84
```

FIGURE 6 Displaying MTU Sizes

- Transport data unit (TDU)

This Sql*Net layer will store data up to this value before transferring to the SDU.

- Session data unit (SDU)

This is the lowest network layer for Sql*Net. Packets sent from Sql*Net to the OS will be no bigger than this size.

For example, take the following settings: SDU=3000, TDU=15000

SQL*Net will store up to 15000 bytes in a buffer (TDU) and send this on the network. The lower network layer (SDU), however, will split this packet up into five packets of 3000 bytes. The 3000-byte packets will be sent to the network controller and be broken up by the MTU into ten 1500 byte packets.

Solaris OE NICs support an ethernet frame MTU of 1500 bytes. Regardless of how large the Oracle SDU is set, Solaris OE will break the message into separate packets. This is not a problem. It is standard practice to increase the Oracle SDU to large values.

*Resolving Sql*Net Waits*

It is good practice to set the Oracle SDU to a multiple of the supported MTU. A good rule of thumb is to use multiples of 1500 for the MTU/TDU. Additionally, it is best to keep the SDU equal to the TDU. Figure 8 shows how SQL*net was configured for a recent Oracle Applications benchmark.

```
#
# listener.ora
PRD =
  (ADDRESS_LIST =
    (ADDRESS= (PROTOCOL= TCP)
      (Host= 18.1.1.2)(Port= 1521)
    )
  )

SID_LIST_PRD =
  (SID_LIST =
    (SID_DESC =
      (SDU = 15000)
      (TDU = 15000)
      (ORACLE_HOME= /d01/app/oracle/product/817)
      (SID_NAME = PRD)
    )
  )
)

#
# tnsnames.ora
PRD =
  (DESCRIPTION_LIST=
    (DESCRIPTION =
      (SDU = 15000)
      (TDU = 15000)
      (ADDRESS = (PROTOCOL = TCP)
        (HOST = 18.1.1.2)(PORT = 1521))
      (CONNECT_DATA = (SID=PRD))
    )
  )
)
```

FIGURE 7 Sample SDU/TDU Settings for SQL*net

Checkpoint Tuning

Cleaning modified buffers is a critical function of an RDBMS. Checkpoints have to complete in a reasonable amount of time and free buffers need to be supplied to active sessions. Oracle 8i introduced some significant changes in order to efficiently clean very large buffer caches.

Oracle 8i introduced the concept of incremental checkpointing as well as multiple database writer processes. These techniques help limit recover and checkpoint time while supplying a continuous stream of clean buffers.

Identifying Checkpoint Problems

Oracle statistics show this problem in the wait events sections of the `report.txt` or `StatsPack free buffer waits` event will be the most dominant wait event. The actual time to complete a checkpoint can be monitored in the `alert.log` file if the `init.ora` parameter `LOG_CHECKPOINTS_TO_ALERT=TRUE`.

Figure 9 shows the wait events section from a `report.txt` where the database could not supply enough clean buffers.

EVENT	TOTWAITS	WAIT	PERTX	TIMEWAIT	AVG_WAIT
free buffer waits	183273	0.5	14647209	79.9	
db file sequential read	4802255	13.1	4691594	.97	
SQL*Net message from client	843754	2.3	3289939	.89	
log file sync	369226	1.0	1984818	5.37	
enqueue	12294	.1	968264	78.75	

FIGURE 8 Free Buffer Waits

Database Writer Processes

The `init.ora` parameter `db_writer_processes` controls the number of database writer processes that are responsible for cleaning buffers. Database writer processes (DBWR) are assigned LRU lists from the default, recycle, and keep buffer pools in a round-robin fashion. To keep an even distribution of work across all DBWR processes, the number of LRUs in each buffer pool should be an integral multiple of the number of database writer processes.

The example in Figure 10 shows three DBWR processes. Each DBWR process is responsible for cleaning one LRU list from each of the three buffer pools. For this example, `db_block_lru_latches=9` and the recycle and keep buffer pools each had three LRU latches.

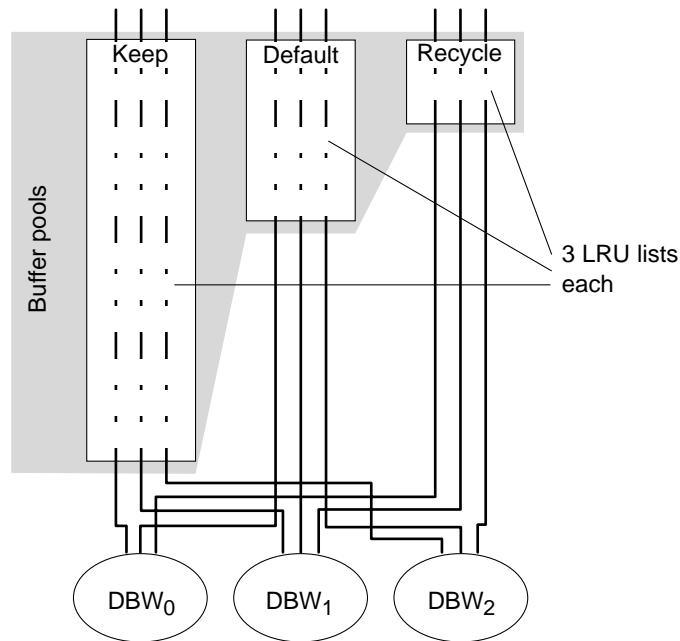


FIGURE 9 Oracle 8i and Beyond Pools, LRUs, and DBWR Processes

Always use `db_writer_processes` *not* `dbwr_io_slaves`. If `dbwr_io_slaves` is set, only one DBWR process is allowed. `dbwr_io_slaves` will be used only to issue IO, which creates a bottleneck because only one DBWR process will be used to scan the LRU lists. IO slaves are only useful for operating systems that do not support kernel asynchronous IO (KAIO). Solaris OE has had support for KAIO for years.

Incremental Checkpoints

The basic idea with incremental checkpointing is to start cleaning buffers *before* a checkpoint occurs. This will reduce checkpoint time and thus the recovery time. Since flushing modified buffers is detached from transaction processing, there is minimal performance impact to using incremental checkpointing.

In Oracle 8*i*, there are two ways to control when incremental checkpointing starts flushing modified buffers.

`db_block_max_dirty_target`

This parameter specifies the maximum number of buffers that are allowed to be modified (dirty). Once the number of dirty buffers exceed this threshold, DBWR processes will start writing dirty buffers to disk, thus lowering the number of buffers that need to be flushed during a checkpoint.

Our experience has shown that setting the `db_block_max_dirty_target` parameter to half the number of `db_block_buffers` is ideal. This setting allowed a checkpoint on a 32-Gigabyte SGA to complete in about 10 minutes after a log switch.

```
db_block_max_dirty_target = db_block_buffers/2
```

If you wish to disable this feature, set `db_block_max_dirty_target=0`.

`db_block_max_dirty_target` is only supported in Oracle 8*i*. In Oracle 9*i*, `db_block_max_dirty_target` has become an Underbar parameter `_db_block_max_dirty_target`.

Note – The `db_block_max_dirty_target` parameter may need to be modified to achieve recovery requirements. Refer to the Oracle Reference Manual for details.

`fast_start_io_target`

This parameter limits the number of IOs necessary for recovery. Oracle keeps track of the number of IOs required to do a fast recovery of the database. Setting this parameter limits recovery time like `db_block_max_dirty_target` but it is triggered by redo log blocks.

If you wish to disable this feature, set `fast_start_io_target = 0`.

`fast_start_io_target` is only supported in Oracle 8*i*. In Oracle 9*i*, `fast_start_io_target` has become an Underbar parameter `_fast_start_io_target`.

In Oracle 9*i*, `fast_start_mttr_target` is the preferred method to limit recovery time. `fast_start_io_target` is defined as the number of seconds that you wish recovery to take.

Data Layout Issues

As with any database configuration, IO must be carefully planned. This article will not go into detailed implementation. There have been volumes written on this subject. A few items that are key to large scale RDBMS implementations are noted.

- Stripe and mirror everywhere (SAME). We called this wide-thin disk striping or plaiding years ago, now Oracle calls it SAME. It is the *same* technique. Instead of partitioning table and index data onto separate disks, it is more beneficial to create large stripe sets and spread data files across all sets.
- Write cache and mirroring for redo logs. Since each user *must* wait upon each commit until the before image information is written to the redo log, it is critical to make this process occur as quickly as possible.

Latch Contention Remedies

There are several Solaris OE and database features that can be used to improve performance when latch contention is the problem. Note that the best fix is to eliminate the need for a latch or create more latches. These remedies were covered in previous sections. This section discusses techniques that show an overall improvement in latch contention.

Processor Sets: `psrset(1M)`

Solaris OE has a feature that allows system administrators to partition CPUs into groups. This is useful to separate critical Oracle processes from the rest of the system.

The `psrset(1M)` command is used to create and administer processor sets. In Solaris 7 OE and beyond, it is possible to disable device interrupts by processor set. This allows for even greater isolation of critical processes.

On multiple projects where network IO and disk IO are heavy, it is beneficial to separate Oracle instances into a separate group and disable interrupts.

Consider the following:

- 24 CPUs
- 32 x FCAL disk controllers
 - 60,000 IO/sec TOTAL
- 3 x QFE Cards
 - 30,000 pkts/sec TOTAL

This system will service interrupts by 35 devices. Each device is mapped to a CPU at startup. With more devices than CPUs, each Oracle user process will be capable of being interrupted. Latching issues will be magnified if this Oracle process is holding a critical latch. To prevent this situation, the instance could be partitioned into a processor set with interrupts disabled.

Table 1 shows 22 CPUs partitioned for Oracle and 2 CPUs for interrupts. This can be done by using the commands in Figure 11.

TABLE 1 Processor Set Partitioning

Process Type	CPU #	Interrupts?
Oracle background processes and listeners	2 through 23	N
Default	0,1	Y

```

$ psrset -c 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
processor set 1 created
$ psrset -f 1 #disables interrupts
$ psrinfo #displays proc info
$ psrset -b <set_id> <pid> #bind pid to a processor set

```

FIGURE 10 Using Processor Sets

Scheduling Enhancements

To optimize for large user counts, it may become necessary to modify the behavior of the scheduler. The goal is to reduce the number of involuntary context switches while a process is holding a critical latch. To keep a process from context switching involuntarily, the time-quanta can be increased.

Sun has created an API that lets the developer put a process in a real-time priority when a process is holding a latch that is critical to the application. Unfortunately, not all versions of Oracle support this feature, but there is another way of accomplishing the same goal.

The dispatch table can be modified with the `dispadm(1M)` command. The dispatch table maps system priorities with time-quanta.

`init.ora: spin_count`

The `init.ora` parameter `spin_count` is used to control how many times an Oracle process spins for a latch before going to sleep. This parameter is notorious for being set too low. In Oracle 6.0, `spin_count` was set to 2000 by default and has never changed.

In a larger CPU environment, it may be more worthwhile to spin, waiting for a process on another CPU to finish, than it is to swap out and lose your context.

Additionally, Oracle has hidden this as an underbar parameter (ie. `_spin_count`) from version to version.

For the UltraSPARC II and UltraSPARC III CPUs, `spin_count` in the range of 8,000 to 20,000 is useful. Be careful, however. If this is set too high, and the work under the latch is long, a process will waste CPU cycles spinning for a latch that it has little probability of obtaining.

References

1. Sun documentation is available at: <http://docs.sun.com/Solaris Tunable Parameters Reference Manual>, part number 806-4015
2. *Optimization of High Volume OLTP Workloads using Very Large Buffer Caches with Oracle 8i and Solaris 7* (presented at SUPERG) Glenn Fawcett - April 1999
3. Oracle Metalink Note 181489.1 on *Tuning Inter-Instance Performance in RAC and OPS*.
4. Oracle 8i and 9i documentation (*Server Reference and Tuning Guide*) <http://technet.oracle.com/>
5. OraPub, Inc. <http://www.orapub.com/>
6. Oracle Articles on StatsPack.
<http://otn.oracle.com/deploy/performance/pdf/statspack.pdf>
http://otn.oracle.com/deploy/performance/pdf/20TUNING_dialeris.pdf
http://otn.oracle.com/deploy/performance/pdf/statspack_tuning_otn_new.pdf
7. *Oracle 8i Internal Services for Waits, Latches, Locks, and Memory* by Steve Adams <http://www.ixora.com.au/>
8. *Optimal Storage Configuration Made Easy* by Juan Loaiza http://otn.oracle.com/deploy/performance/pdf/opt_storage_conf.pdf

Author

Glenn Fawcett has twelve years experience tuning large-scale database systems and is part of the Strategic Applications Engineering Group at Sun. Glenn and his group have the best hands-on experience with large servers as they are dedicated to understanding how to improve overall performance for various software packages and applications. Glenn was the project lead for multiple performance benchmarks with Oracle for OLTP and DSS workloads. Glenn is also a regular speaker at the Sun Users Performance Group conference *SUPerG*.

Acknowledgments

I would like to thank the Sun SAE group, Akiko Marti (customer benchmarking), Sun Performance and Applications Engineering Group, Sun Market Development Engineering, and various Sun customers for providing me insights into the unique use of this technology.