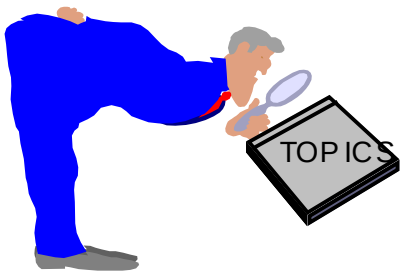
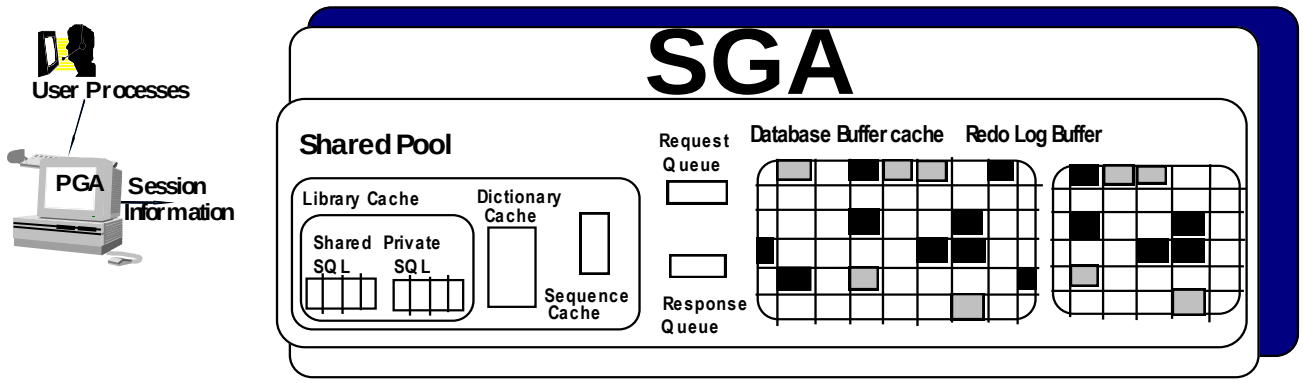
TOPIC OBJECTIVES



- **Oracle Memory Structures**
- **Database Buffer Tuning**
- **Data Dictionary Tuning**
- **Redo Log Tuning**
- **Rollback Segment Tuning**

MEMORY STRUCTURES

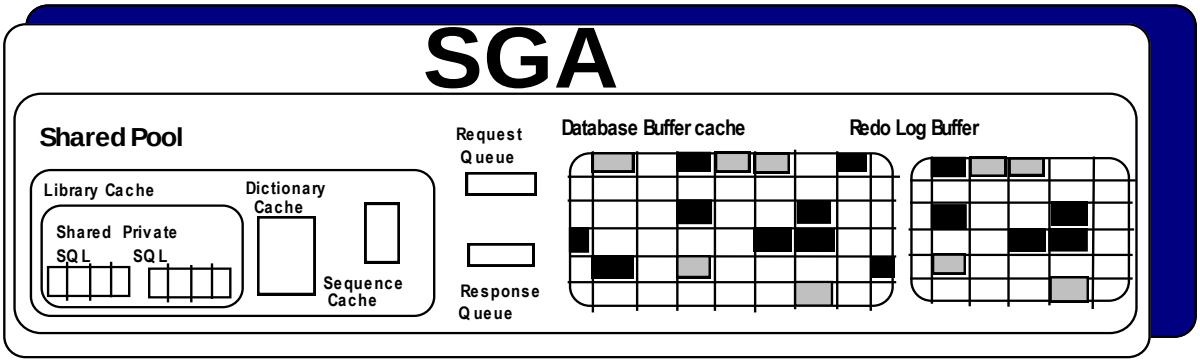


ORACLE uses many memory structures; four important memory structures are:

- Software code Area - Portions of memory are used to store code that is being or may be executed are called software code areas.
- SGA (System Global Area/ Shared Global Area)
- PGA (Program Global Area)
- Sort Area

NOTE: For better performance, store as much data as possible in every memory structure.

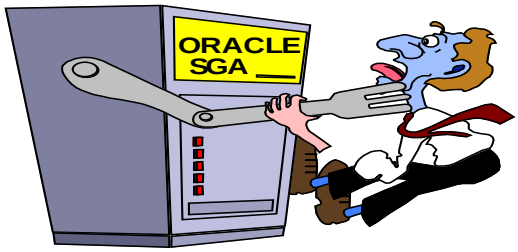
SHARED GLOBAL AREA (SGA)



The SGA is a location in shared memory that contains memory structures that hold data and control information for one instance which is frequently accessed by ORACLE processes.

- For optimal performance, the entire SGA should always be contained in real memory (not virtual memory) to avoid the SGA being paged or swapped
- Be careful not to create too large of an SGA which could cause paging. The disadvantage of excessive paging significantly outweighs the advantage of a large SGA, except in some cases of paging to expanded memory
- Part of the SGA is called fixed and contains information that the background processes need
- SGA memory usage is displayed at STARTUP or with the SQLPLUS command SHOW SGA

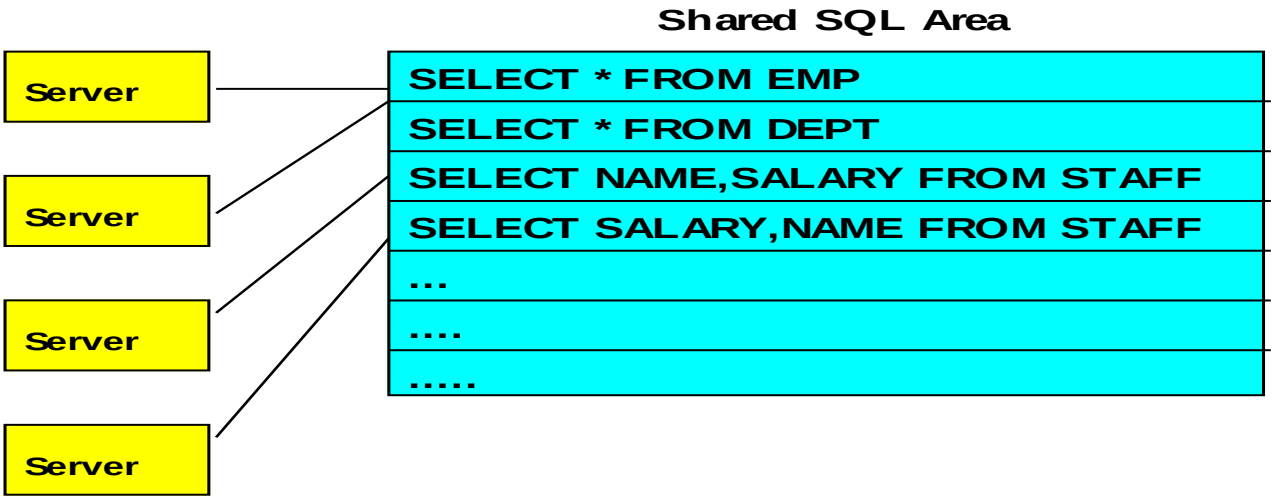
MORE ON SGA



I need more memory

- The size of the SGA is defined and allocated at STARTUP.
- The SGA is released during SHUTDOWN
- The size is mainly affected by the parameters:
  - DB_BLOCK_SIZE 2k – 32k) – 64k for 64 bit Unix systems
    Size of data blocks
    Typical values (2k – 4k..8k..16k, 32k.)
    Block size is established at CREATE DATABASE time
    Affects FREELISTS storage parameter for tables and indexes
- DB_CACHE_SIZE
  - Size of default buffer cached in SGA memory.
  - This parameter probably has the greatest impact on performance
- LOG_BUFFER ([4* DB_BLOCK_SIZE]. OS dependent)
  - Size in bytes of the SGA's redo log buffer
  - A size of 65KB or greater is reasonable for very active systems
- SHARED_POOL_SIZE [3.5MB..400GB]..OS dependent)
  - Size of shared pool in bytes

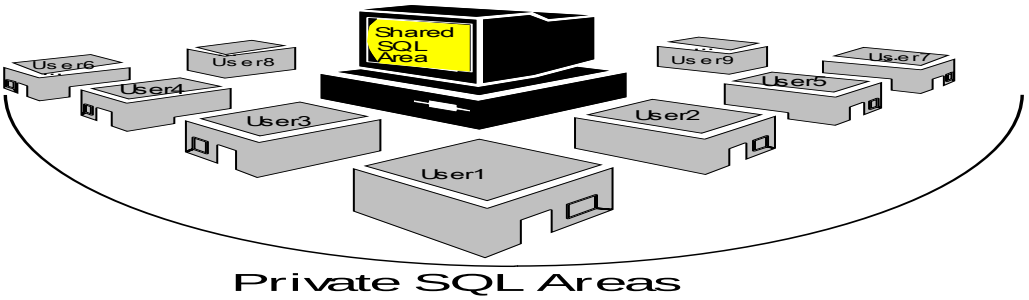
SHARED SQL AREAS



The shared pool is part of the SGA that contains shared memory constructs. Some such memory constructs are the library cache (shared SQL areas and private SQL areas), and the data dictionary cache.

- The shared areas are used to process each SQL statement regardless of its origin
- The shared pool also retains compiled anonymous and named blocks (stored procedures) and packages
- The shared SQL area contains the execution plan and the parse tree for a single SQL statement and is shared by multiple local or distributed applications that issue identical SQL statements.
- ORACLE evaluates user and application initiated SQL statements and recursive SQL statement initiated by ORACLE for uniqueness. If the submitted SQL statements are identical in case, character and spacing, reference the same object in the same schema, bind variables match in name and type, and the same optimization approach and optimization goal are used, a shared SQL area is used

PRIVATE SQL AREAS



Private SQL Areas contain run-time buffers and binding information.

- Every user that issues a SQL statement has their own private SQL area, and may have their own or share an SQL shared area.

- The size of the runtime area is dependent upon the:
  - Type of SQL (DML or SELECT)
  - Complexity of the SQL statement
  - Row size
  - Runtime area released after DML executed

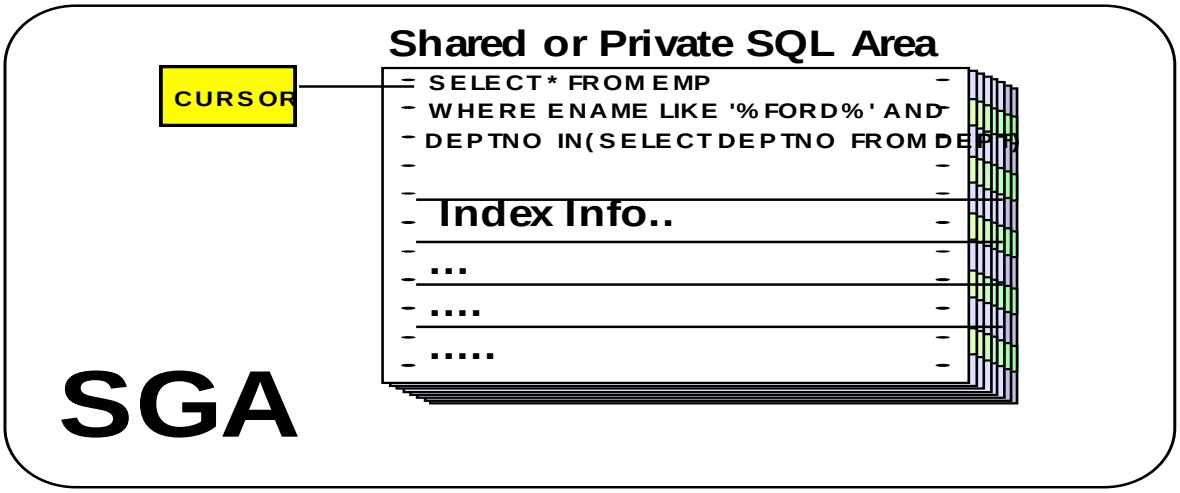
- When a cursor is closed for a SQL statement, the corresponding private SQL area is released. Close all unneeded open cursors

- To identify SQL contention and identify how often an item was executed to the number of reloads (how often a miss occurred) use the following formula and query:

  ratio = (reloads / pins) * 100 - If the result is greater than 1 increase the size of the SHARED_POOL_SIZE in the spfile file.
  See librcach.sql

  SELECT SUM(PINS) PINS, SUM(RELOADS) RELOADS.
  SUM(PINS-RELOADS)/SUM(PINS) * 100 " Cache Hit Ratio"
  FROM V$LIBRARYCACHE;

CURSORS AND CONTEXT AREAS



Cursors are names for, or pointers to, memory structures (context areas: shared and private SQL areas) in the SGA where parsed statements and their information are kept

- Cursors has a location in memory that contains the information required to process a SQL command. The location in memory is identified by a cursor name.

- A session can have up to OPEN_CURSORS concurrent cursors

- Cursor areas are defined by three parameters

  - OPEN_CURSORS (default = 50; range : 5-255)

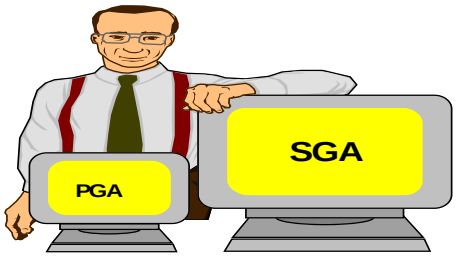
ALTER SYSTEM RESET OPEN_CURSORS SCOPE=SPFILE sid='RACDB1';

  - CURSOR SHARING
    Values are EXACT, SIMILAR or FORCE
    SIMILAR is used for DSS systems where the same SQL Statement is used many times

.

  - SESSION_CACHED_CURSORS
    This initialization parameter will cache up to the number session cursors and improves performance of repeated Parse calls to the same SQL statements.

MANAGING SQL AREAS

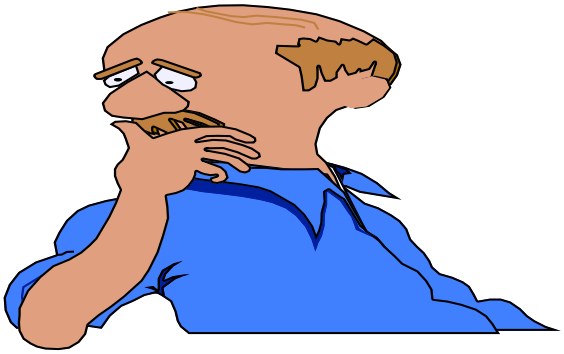


The user process is responsible for the management of private SQL areas through the application that is being used.

- The number of private SQL areas is limited by OPEN_CURSORS
- SHARED_POOL_SIZE controls the amount of memory allocated for the shared SQL area
- If there is not enough contiguous space in the shared pool, ORACLE can release objects from the pool using a modified LRU algorithm until there is enough free space
  - Increase SHARED_POOL_SIZE if the multi-threaded server id used because the session information is contained in the shared pool
  - Query V$SESSTAT when the application is running in multi-threaded server. V$SESSTAT contains session information ('session memory', and 'max session memory') that can help you determine if more memory is needed.
  - Be careful not to allocate too much memory so that paging and swapping occurs.
  - Reduce Library cache misses by:
    - Increasing memory for the library cache until V$LIBRARYCACHE.RELOADS is close to 0.
    - Write identical SQL statements

DO'S AND DON'TS FOR MANAGING SQL AREAS

How do I manage my SQL AREAS?



- Do not allow users to change the optimization approach and optimization goals for their session

- Standardize:

  - Use bind variables instead of constants in statements
  - Naming conventions for bind variables
  - SQL command and PL/SQL block construction, i.e., indenting, spacing
  - Use stored procedures whenever possible

- CURSOR_SPACE_FOR_TIME

  - FALSE (default) allows ORACLE to deallocate space held by a statement in the shared pool even if application cursors using the statement is still open

  - TRUE requests ORACLE to wait until all application cursors using a statement is closed. This should only be used if the shared pool can hold all open cursor data.

TUNING THE SHARED POOL RESERVE SPACE

SQL & PL/SQL

Some operations in the SHARED POOL AREA require large contiguous memory allocations, such as compiling PL/SQL or temporarily storing java objects. If you have these types of operations, Oracle recommends configuring the RESERVED POOL by setting the spfile parameter SHARED_POOL_RESERVED_SIZE to 10 percent of the SHARED_POOL_SIZE after tuning the SHARED_POOL_SIZE. Large chunks (PL/SQL routines particularly) are allocated by:

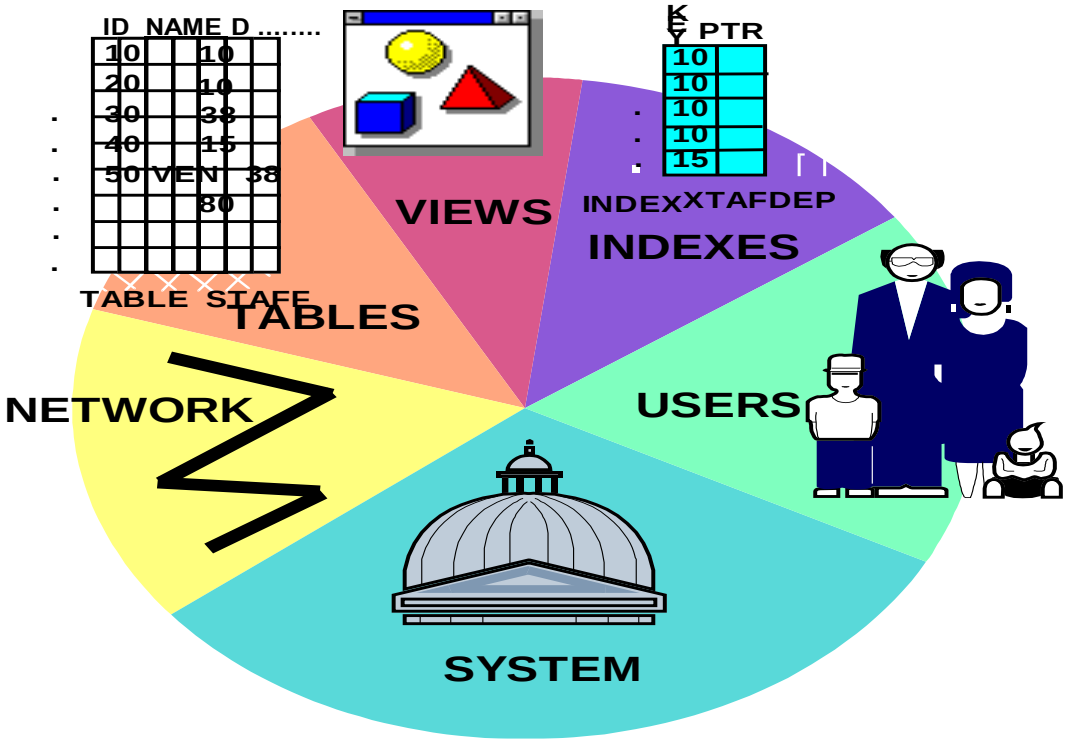
1. from the unreserved section of the shared pool

2. If there's not enough space there and the allocation is large, look in the reserved pool for available space

3. If there's still not enough free space, begin freeing memory chunks, then retry the unreserved and reserved.

Note. The primary purpose of this parameter is to reduce fragmentation in the SHARED_POOL_SIZE. To identify performance issues look at V$SHARED_POOL_RESERVED and expand the reserved pool and possibly the shared pool if REQUEST_MISSES and REQUEST_FAILURES in this view are greater than 0.

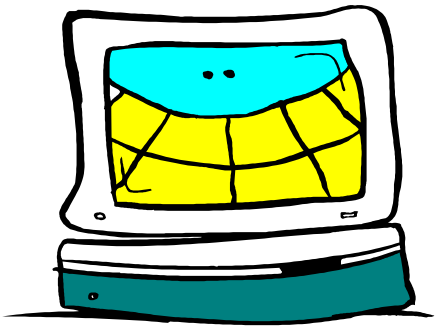
DATA DICTIONARY CACHE



The Data dictionary cache, or row cache, contains information about the database. Some of the information refers to tables, views, columns, types, users, and user privileges of all ORACLE users.

- The data dictionary cache is comprised of several caches, each holding information for a particular database object
- The data dictionary cache is shared by all users
- The size of the data dictionary cache is dependent on SHARED_POOL_SIZE. To increase the memory available to the cache increase this parameter
- Unlike ORACLE earlier versions, there is only one parameter that can change the size of all the data dictionary caches

MORE ON DATA DICTIONARY CACHE



***THAT IS A "* GOOD RUNNING " SQL STATEMENT**

SELECT * FROM V$ROWCACHE;

- A data dictionary cache miss is more costly than a buffer cache miss
- Dictionary data is held in memory longer than library cache data
- Tune before buffer cache and after context area
- Query V$ROWCACHE to show cache hits and gets
- The ratio of total hits (get misses) and total gets should be less than 10% or 15%. Use the following formula for the hit ratio:

  SQL> select (SUM(GETS – GETMISSES)) / SUM (GETS)
  “Dictionary Cache Hit Ratio”
  from V$ROWCACHE

  Increase the SHARED_POOL_SIZE parameter if greater than 10%.
  See dictcach.sql

SHARED POOL MEMORY ALLOCATION

Steps for allocating memory

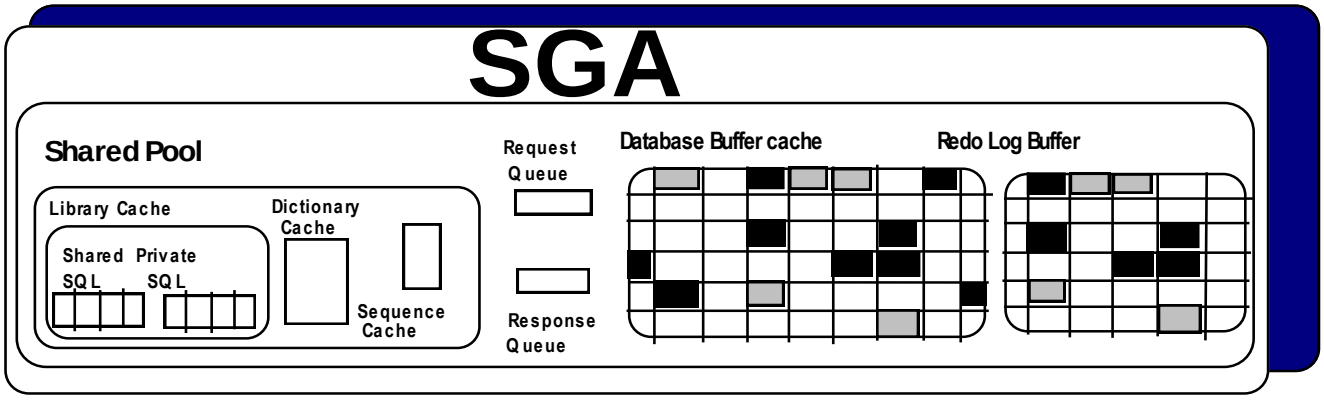
1. Allocate Private SQL Areas
2. Check if issued statement is in a Shared SQL Area
3. If found, execute the Shared SQL
4. If NOT found, allocate a new contiguous area in the Shared Pool

- Entries in the shared pool remain until they are flushed using the modified LRU algorithm. LRU will flush when there is not enough contiguous space to satisfy incoming statement

- The LRU algorithm keeps unused entries in the shared pool, even if the process that created the entry is terminated, until space is required for a new entry

- If an object is modified, dropped, or created, or a procedure or package is recompiled, the shared SQL area that is dependent upon these objects is marked invalid. ORACLE must reparse these SQL statements to regenerate a valid shared SQL area when the cursors that corresponds to the invalid shared SQL area are executed after the object was altered or compiled

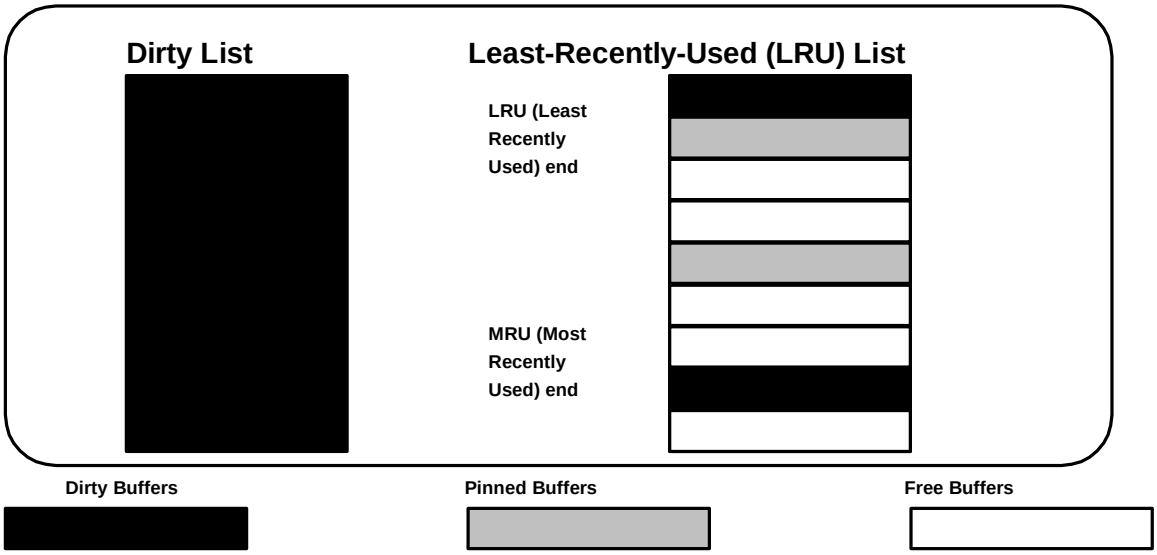
CACHE



A cache is a memory location that contains data also stored on disk. Three important caches are:

- Library Cache
- Data Dictionary Cache
- Database Buffer Cache

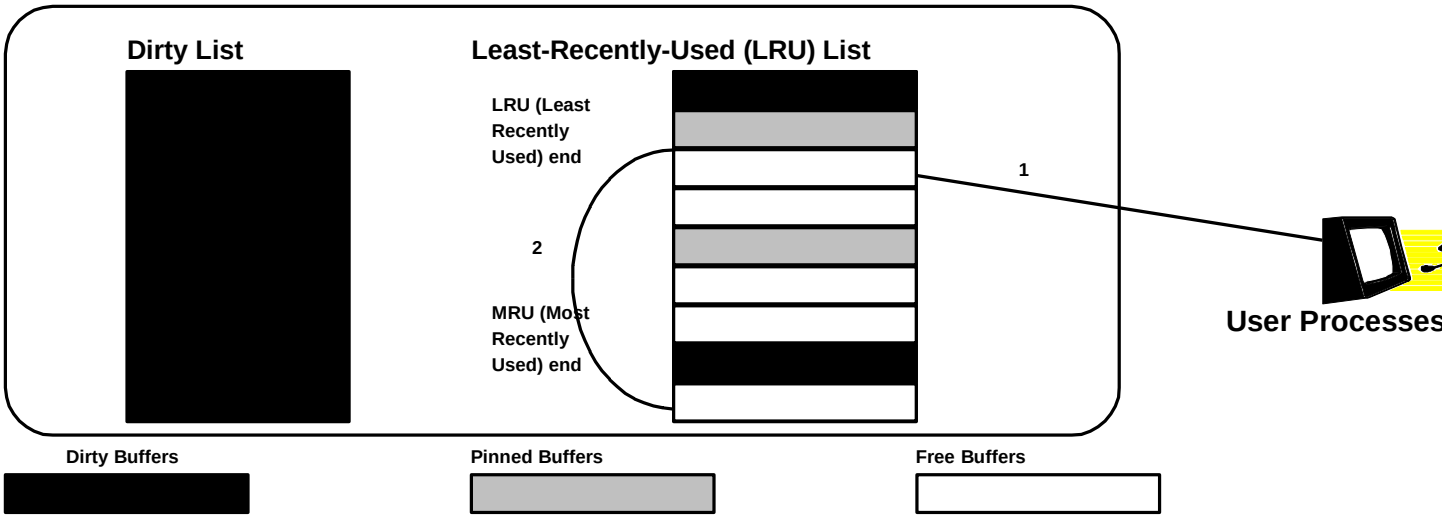
DATABASE BUFFER CACHE



The Database Buffer Cache

- Holds copies of database blocks
- Each buffer holds a single ORACLE block
- A total of 5 buffers each with a different block size can be used
- Buffer cache is organized in two lists:
  - dirty list
  - Least-recently-used (LRU) list
- The dirty list contains buffers that have been modified but not yet written to disk. These buffers are called buffer dirty buffers.
- The LRU list contains dirty buffers that have not yet been moved to the dirty list, pinned buffers, and free buffers.
  - Pinned buffers are buffers that are being accessed (modified)
  - Free buffers are buffers that have not been modified and are available for use
- Number of buffers in the cache is determined by the spfile parameter DB_CACHE_SIZE. This is the DEFAULT size for the SYSTEM tablespace.

MORE ABOUT DATABASE BUFFERS

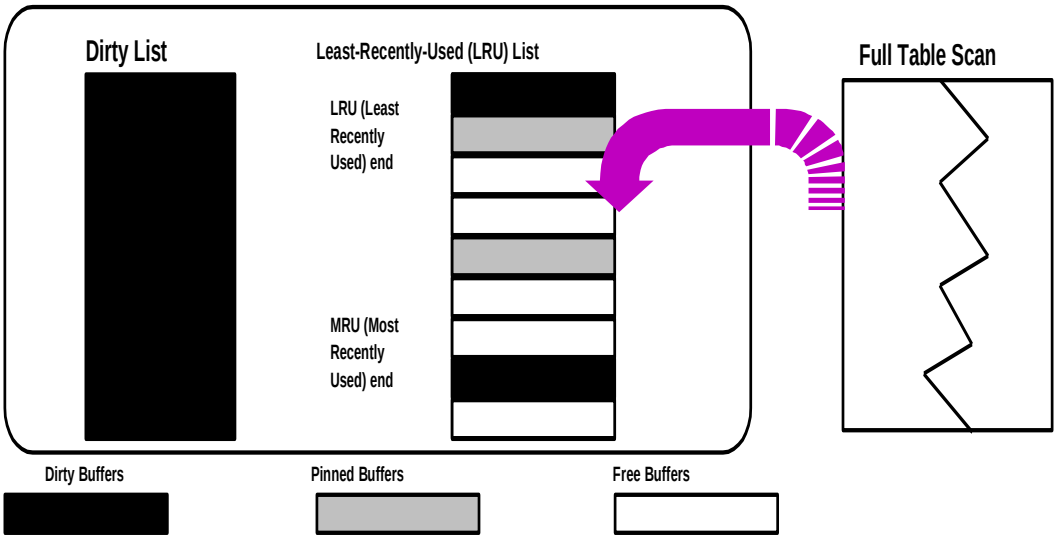


When a user process has a request to access a block stored in the cache, the process moves the buffer that contains the block to the most recently-used (MRU) end of the LRU list. The user process can then access the buffer

If the user process has a request to access a block not in the buffer cache, the process:

- Uses a LRU algorithm to search the LRU end of the LRU list for a free buffer until:
  - a free buffer is found
  - DB_BLOCK_MAX_SCAN_CNT buffers have been searched without locating a free buffer. In this case, the user process stops searching and informs DBWR to write some dirty buffers to disk.
- During the search, if the user process locates a dirty buffer, it transfers the buffer to the dirty list and resumes searching
- When a free buffer is found, the process reads the block into the buffer and moves it to the MRU end of the LRU list.

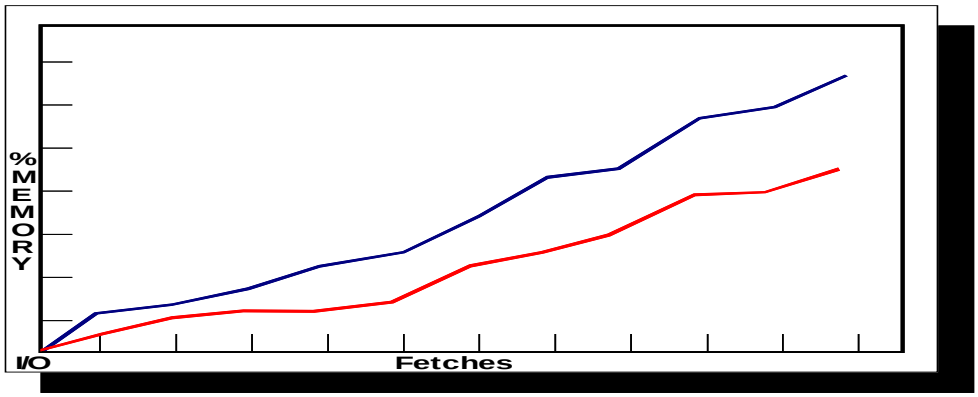
DATABASE BUFFER CACHE



When a process is performing a full table scan of any large table, the blocks are placed on the LRU end of the LRU list. Because the data from a large full table scan is generally needed briefly, the blocks should be flushed as soon as possible from the list to allow more than frequently used blocks in the cache.

- The database buffer cache has two main performance advantages:
  - reduced disk I/O by user process
  - Buffer cache is shared
- Large cache sizes reduce writes and may take too much memory which would cause paging or swapping
- Increased cache sizes increase the percentage of cache hits

STATISTICS FOR THE DATABASE BUFFER CACHE



Statistics are collected for:

- Logical reads: total number of data requests (disk and memory)
- Physical reads: total number of data requests from disk
- Hit ratio: percentage of total data requests satisfied by memory access (logical reads - physical reads) / logical reads

A cache hit is when a user process accesses data stored in a cache

A cache miss is when the user process requests data that is not in the cache and must copy the data from the data files to the cache before processing it.

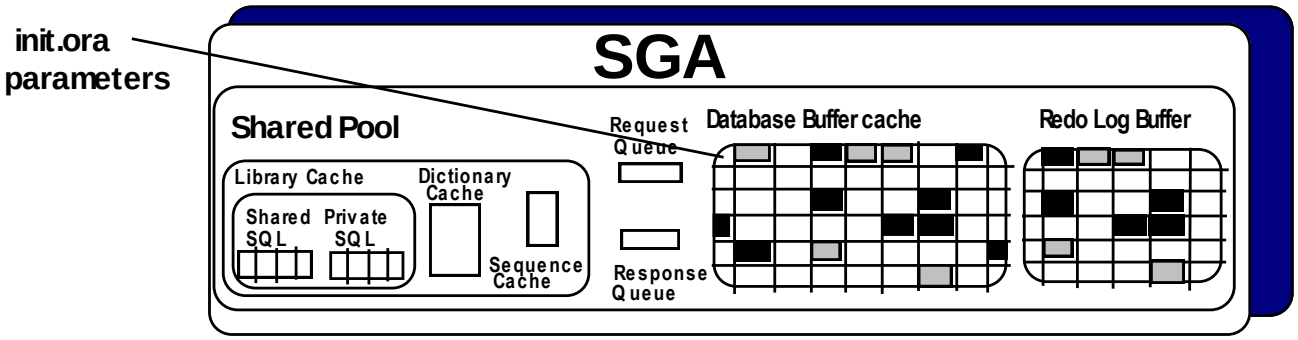
The hit rate within the database buffer cache should be very high -- greater than 70%. To determine the rate, execute the following query and use the algorithm:

select substr(name,1,50, value from v$sysstat where
name in('consistent gets', 'db block gets', 'physical reads');

hit ratio = 1 - (physical reads / (db block gets + consistent gets))

Increase the DB_BLOCK_BUFFERS if less than 70 percent.

DATABASE BUFFER CACHE PARAMETERS



DB_xk_CACHE_SIZE is a dynamic parameter whose value determines the Size of the buffer and the block size in the buffer cache of the SGA. Oracle9i supports multiple block sizes in the same database. The block size may be from 2k – 32k.

- Large cache sizes reduce writes and may require much memory and cause swapping or paging
- A rough approximation of the minimum number of buffers should be the number of database blocks that an application uses most frequently. those blocks should include rollback segments, indexes, and tables
- After creating a database the default block size set by DB_BLOCK_SIZE. The default buffer size is set through DB_CACHE_SIZE.
- Multiple block sizes can be established by creating a non standard block size buffer cache. This is done by the following commands:

SQL> Alter SYSTEM SET DB_32K_cache_SIZE = 40M

SQL> Create tablespace big32tbs
Datafile ‘c:\warehouse\big32tbs.dbf’ size = 20m blocksize 32k;

BUFFER CACHE SIZE INFORMATION

Here is the buffer advice!!

When first configuring an Oracle instance, there's really no way to know an accurate value for DB_CACHE_SIZE. If it is set too low than I/O suffers, too high and memory is unused. Oracle provides an spfile parameter called DB_CACHE_ADVICE which when set on (dynamically or through spfile) collects statistics about cache utilization and projects the physical I/O for 20 different cache sizes, ranging from 10 percent of the current cache size to 200 percent. Here is how the parameter is used:

DB_CACHE_ADVICE

- **OFF** disables advice statistics gathering and does not allocate memory from the shared pool. If the value was ON it deallocates the memory.

- **READY** If set at instance startup, the shared pool memory is preallocated but statistics are not gathered. If altered while the instance is running, the memory is allocated.

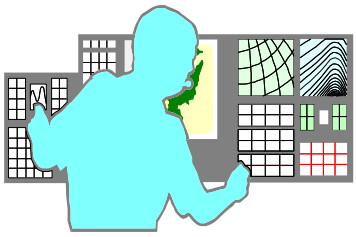
- **ON** Memory is allocated and statistics are gathered. This causes a small increase in CPU utilization.

To see the information on I/O usage:

```
Select SIZE_FOR_ESTIMATE "Cache Size (mb)", BUFFERS_FOR_ESTIMATE "buffers",
        ESTD_PHYSICAL_READ_FACTOR "Read Factor",
        ESTD_PHYSICAL_READS "Estimated Reads"
From V$DB_CACHE_ADVICE
ORDER by 2
```

Note: The above formula allows you to identify the best DB_CACHE_SIZE memory by identifying the amount of REDUCED READS per increase MEMORY ALLOCATION.

CONFIGURING DBWR I/O SLAVES

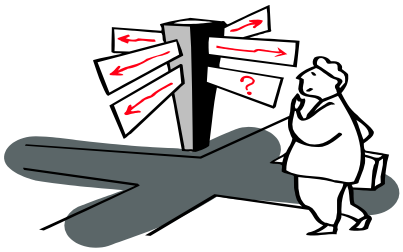


Configure DBW0 I/O slaves by setting an integer value for DBWR_IO_SLAVES. Asynchronous I/O (DISK_ASYNCH_IO) does not have to be turned off. If asynchronous I/O is enabled and you configure I/O slaves, the I/O slaves will use asynchronous I/O. This is especially helpful in database environments with very large I/O throughput.

I/O slaves for DBW0 are allocated when:

1. DBW0 looks for an idle I/O slave; if one is available, it will be used
2. If there are no idle I/O slaves, the DBW0 process spawns one
3. If the maximum of I/O slaves has been reached, determined by the value of DBWR_IO_SLAVES, DBW0 waits and then tries to find an idle I/O slave.
4. DBW0 continues to gather dirty buffers in a batch and prepare them for write.
5. The I/O slave is called to perform the write on behalf of DBW0.

CONFIGURING MULTIPLE DBW PROCESSES



Symmetric multiprocessing (SMP) systems

An instance is configured with one database writer process (DBW0) by default. Multiple DBWn process can be configured by setting the DB_WRITER_PROCESES Initialization parameter to a value greater than 1 and less than or equal to 10. A maximum of 10 processes, DBW0 through DBW9, can be configured. Because DBWn processes gather dirty buffers and then write them to disk, configuring multiple DBWnn processes is more effective than using an equivalent number of I/O slaves with one DBWR process. This parallelization of DBWn processes is useful on symmetric multiprocessing (SMP) systems with many central processing units (CPUs). Multiple DBWn processes cannot be used concurrently with I/O slaves.

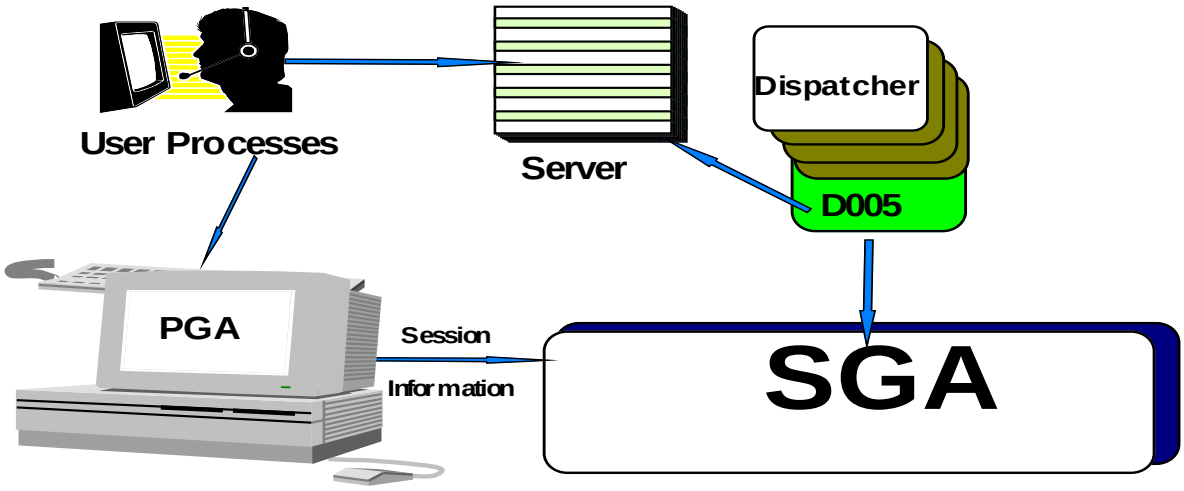
Tuning DBWn processes. The V$SYSTEM_EVENT view is used to determine if you need to configure multiple DBWn processes. For instance:

```
Select EVENT, TOTAL_WAITS
From V$SYSTEM_EVENT
Where EVENT = ‘free buffer waits’
```

EVENT	TOTAL_WAITS
free buffer waits	38

A high number of TOTAL_WAITS indicate that increasing the number of DBWn processes will improve performance.

SINGLE AND MULTI-PROCESS SYSTEMS

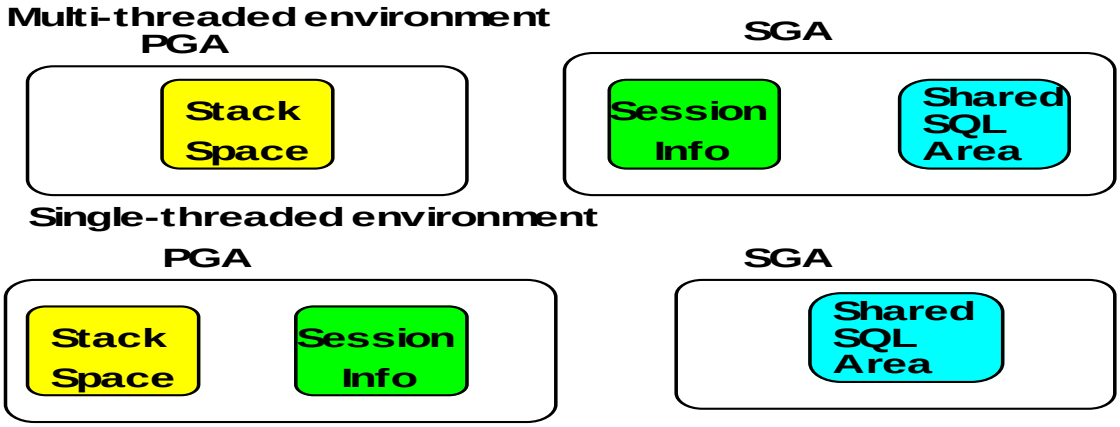


Each server and background process has a memory region that contains data and control information. This memory region is a Program Global Area (PGA).

In multi-process systems, processes can be categorized into two groups:

- User processes
- ORACLE processes (two types)
  - Server processes handle the requests of the user process; parse and execute SQL statements, read data blocks from disk into the database buffers, and return results to the application.
  - Background processes accommodate many users and maximizes performance.
- Communication between the user and ORACLE processes is accomplished through a mechanism called the Program Interface.

USER GLOBAL AREA



The USER GLOBAL AREA (UGA) contains:

- Stack space - session variables, arrays and other information
- Session information - data on user session information, i.e. private SQL areas and cursor state.

In a shared server environment, the user session data and the cursor state are moved into the shared pool (see multi-threaded environment above)

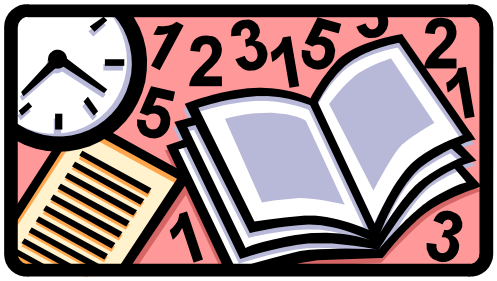
In a dedicated server environment, the user session data and cursor state space remain in the PGA regardless of server configuration.

The PGA is a non-shareable part of memory that can be written to or read from. The size of the PGA is operating system dependent and static. Three spfile parameters that effect the PGA size are:

- OPEN_LINKS
- DB_FILES
- LOG_FILES

Once connected a user will never run out of PGA memory, either there is or there's not enough space in the first place.

UGA/PGA STATISTICS



Where do I get session statistics?

The V$SESSTAT, V$MYSTAT, and V$STATNAME views are joined to report on session UGA memory usage. V$MYSTAT shows the information for the current session; V$SESSTAT provides the information for all sessions. For example to look at current total session memory:

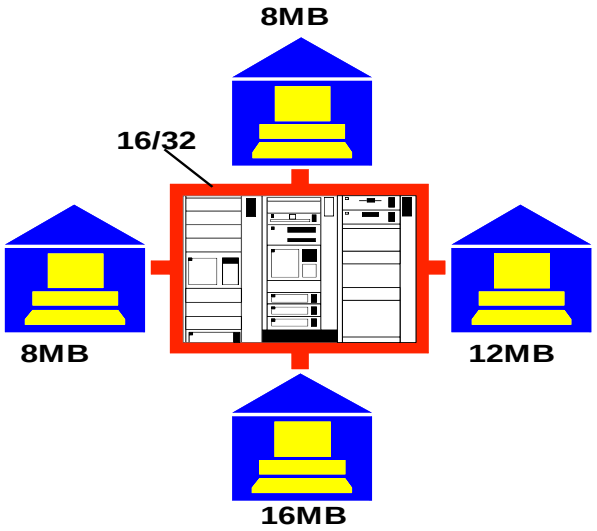
```
SQL>  select SUM(VALUE) “Total Session UGA memory”
         From V$sesstat A, V$STATNAME B
         Where NAME = ‘session uga memory’     and
         A.STATISTIC# = B.STATISTIC#
```

To look at the maximum session memory used:

```
 SQL>  select SUM(VALUE) “Total Session UGA memory MAX”
         From V$sesstat A, V$STATNAME B
         Where NAME = ‘session uga memory max’     and
         A.STATISTIC# = B.STATISTIC#
```

Somewhere between the two values produced will be the optimal increase in the share pool area.

INFORMATION ON SHARED SERVERS



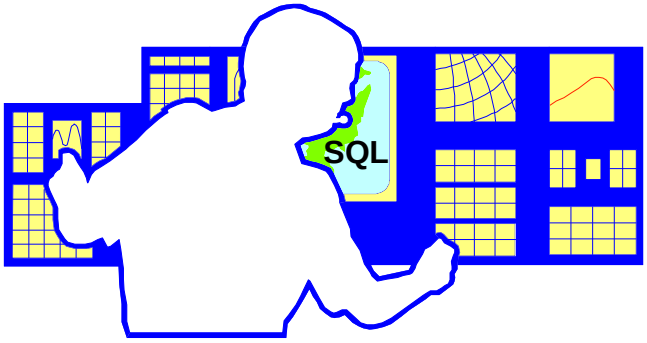
The V$SHARED_SERVER_MONITOR view provides a starting point to see what is happening with the SHARED SERVER environment.

```
SQL>  SELECT maximum_connects,
                  maximum_sessions,
                  servers_started,
                  servers_terminated,
                  servers_highwater
            from V$SHARED_SERVER_MONITOR
```

The columns provide the following information since the instance started:

- MAXIMUM_CONNECTIONS – The highest number of virtual circuits (user connection) in used at one time.
- MAXIMUM_SESSIONS – The highest number of shared server sessions in use at one time
- SERVERS_STARTED – The total number of shared servers started. Excludes system generated servers.
- SERVERS_TERMINATED – The total number of shared servers stopped
- SERVER_HIGHWATER – Highest number of shared server running.

MONITORING DISPATCHERS



How many active dispatchers do we have?

A DBA can monitor dispatcher activity by querying V$SHARED_SERVER_MONITOR, V$DISPATCHER, V$QUEUE, and V$DISPATCHER_RATE,.

- V$DISPATCHER – gathers performance statistics
- V$QUEUE – identifies queue wait time
- V$DISPATCHER_RATE – analyzes detailed dispatcher events

The DBA can use V$DISPATCHER to identify wait time for dispatchers:

```
SQL> SELECT NETWORK "PROTOCOL", SUM(BUSY)/SUM(BUSY) + SUM(IDLE))
        "TOTAL BUSY RATE"
        FROM V$DISPATCHER
        GROUP BY NETWORK

PROTOCOL                                                    TOTAL BUSY RATE
----------------------------------------------------------- ---------------
(ADDRESS=(PROTOCL=tcp) HOST=packnet02) (Port=1035)                .0173043
```

This query indicates the total busy rate for the dispatcher processes, grouped by Protocol. If the rate is greater than .5 it indicates the dispatcher processes are busy More than 50% of the time. Add more dispatchers.

DISPATCHER PERFORMANCE

I'm waiting for a Dispatcher

The DBA can examine several views retrieve information on DISPATCHER activity. Querying V$QUEUE provides information on users waiting for dispatchers:

```
SQL>  SELECT decode(SUM(TOTALQ),0,"No Responses", SUM(WAIT)/SUM(TOTALQ))
          "Average Wait Time"
          FROM V$QUEUE Q, V$DISPATCHER D
          WHERE q.type = 'DISPATCHER'    AND
                    q.paddr = d.paddr
```

NOTE: The average wait time is in hundredths of seconds. If the wait time increases consider adding more dispatchers.

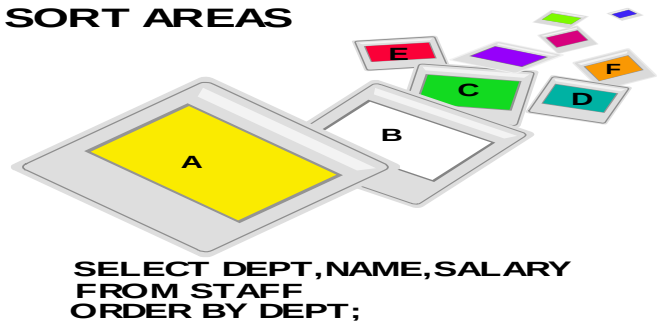
The DBA can also query V$DISPATCHER rate to further analyze statistical deviations Among dispatcher users. You can evaluate dispatcher performance by comparing current columns (anything with CUR_) to maximum value columns (anything with MAX_).:

```
SQL>  SELECT  CURRENT_IN_CONNECT_RATE, MAXIMUM_IN_CONNECT_RATE
          FROM V$DISPATCHER_RATE
```

NOTE: If the values of current and maximum are close than consider adding more Dispatchers.

```
SQL>  ALTER system set dispatchers = '(PROTOCOL=TCP) (DISPATCHERS = 5)'
```

SORT PROCESSES FOR 8i



A SORT area is a memory location in the user process that ORACLE uses to sort data when operations like ORDER BY, GROUP BY, DISTINCT, and CREATE INDEX are used.

The SORT AREA size is defined by SORT_AREA_SIZE bytes parameter.

The SORT AREA can grow large enough to allow the data to be sorted. If the data does not fit in memory, the data is divided into smaller pieces that do fit. These pieces are sorted into runs. finally ORACLE merges the runs to create the final result. To see how often a sort has too little memory use the following script.
See sorthit.sql

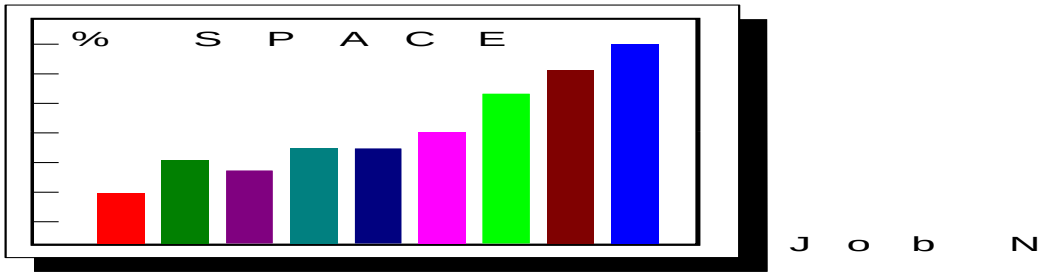
```
select (sum(decode(name,'sorts (memory)',value,0)) /
(sum(decode(name,'sorts (memory)',value,0)) +
sum (decode(name,'sorts (disk)', value,0)))) *100
memory_hit_ratio
from V$sysstat
where name like 'sort%'
```

Or to track the number of in-memory and to-disk sorts:

```
select name,value
from V$sysstat
where name like '%sort%'
```

NAME	VALUE
sorts (memory)	2657
sorts (disk)	3
sorts (rows)	5495

SORT AREAS FOR ORACLE



When the SORT AREA is not used, ORACLE may decide to decrease the size of the SORT AREA by writing some of the data to a temporary segment and then releasing the portion of the sort area that contained the data. ORACLE decreases the SORT AREA size to SORT_AREA_RETAINED_SIZE. To track sorts run:

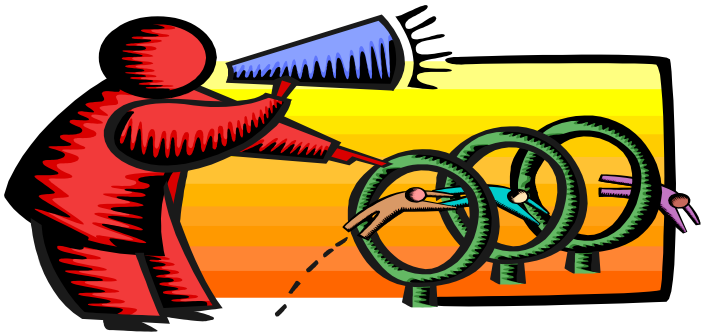
- Sort statistics, which show the number of sorts that were and were not performed entirely in memory
- Whenever possible create an index by loading the rows into the table in ascending order of the indexed column values using the NOSORT option of the CREATE INDEX statement.
- Large SORT AREAS are not free. Large SORT AREAS reduce the amount of memory available for PL/SQL and private SQL areas and may cause paging.

To determine the number of sort performed in Memory:

```
SELECT 100*(A.VALUE – B.VALUE) / (A.VALUE)  "In-memory Sort Ratio"

FROM V$SYSSTAT A, V$SYSSTAT B

WHERE A.NAME = ‘sorts (memory)’  and

        B.NAME = ‘sorts (disk)’
```

In-memory Sort Ratio
99.8876

ORACLE NEW SORT PARAMETERS



Announcing automated sorting with dedicated servers

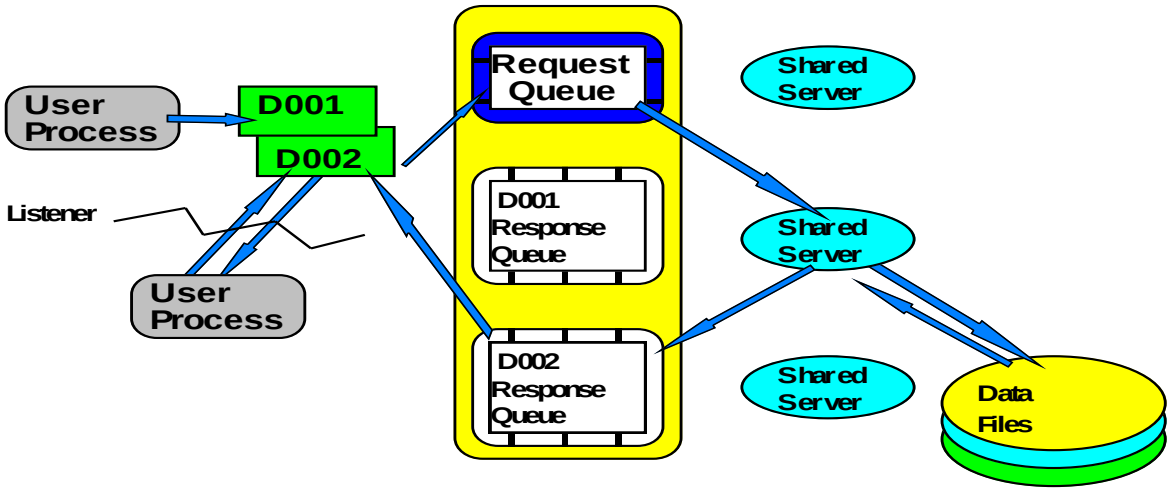
- PGA_AGGREGATE_TARGET
- WORKAREA_SIZE_POLICY

The initialization parameter PGA_AGGREGATE_TARGET specifies the target aggregate PGA memory available to all dedicated server processes. Set this parameter value between 10mb and 4000 GB. When WORKAREA_SIZE_POLICY is set to AUTO, Oracle adapts the size of the sort areas to meet private memory needs.

When PGA_AGGREGATE_TARGET is defined, setting the WORKAREA_SIZE_POLICY to AUTO causes work areas used by memory intensive operators to be sized automatically. The size of the work areas is based on the PGA memory used by the system, the PGA_AGGREGATE_TARGET and the requirement of each individual operator.

Look at V$PGASTAT view to diagnose PGA memory utilization or look at statspack report on PGA Memory Stats.

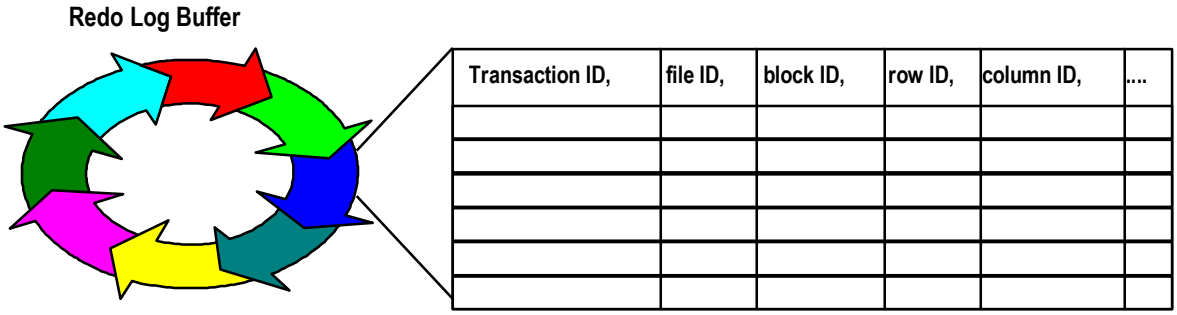
SHARED SERVER CONCEPT



- To use the shared server, a user process must use SQL*NET, no matter If the user and server are on the same machine
- The number of total running shared servers is between the two parameters MTS_SERVERS and MTS_MAX_SERVERS
- ORACLE dynamically creates or removes shared servers based on the length of the request queue
- Artificial deadlock can occur with a limited number of shared servers

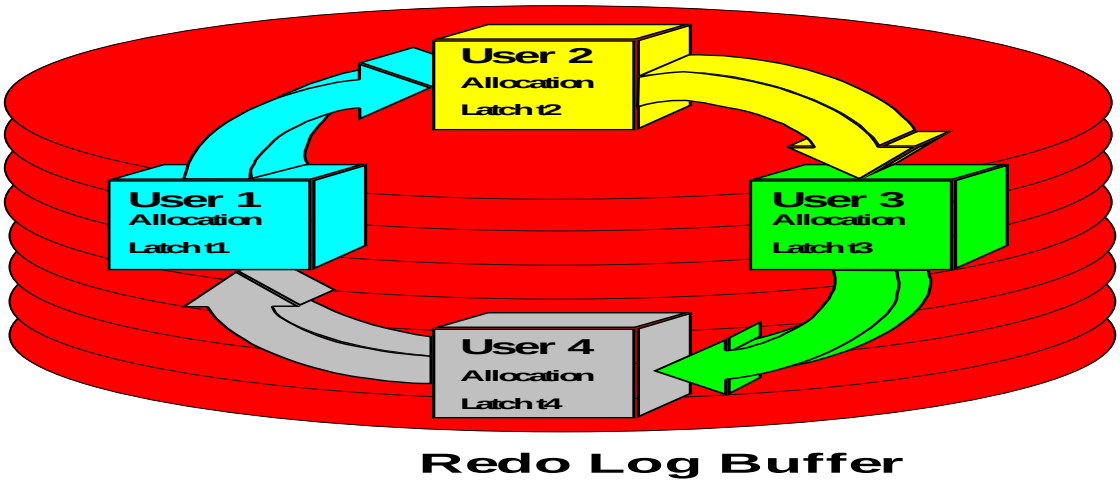
NOTE: When artificial deadlock is detected, new shared servers are created by ORACLE up to the maximum spfile (MTS_MAX_SERVERS) to increase the available system resources to remove the deadlock. If the maximum number of shared servers have been created and deadlock still exists, the DBA must manually remove users from the system until the deadlock is resolved.

REDO LOG BUFFER ENTRIES



- The log entries are used for database recovery and for the reconstruction of data in the database
- The background process Log Write (LGWR) writes the log buffer to the current active on-line redo log file group when either a transaction is committed, or when the buffer fills. Once the entries have been written to a redo log file group, other server processes can the copy new entries over the entries in the redo log buffer that have been written to the redo log file
- An entry contains a transaction ID, file ID, block ID, row ID, column ID and other transaction information for INSERT, UPDATE, DELETE, CREATE, ALTER and DROP commands

MORE ABOUT THE REDO LOG BUFFER



LOG_BUFFER determines the size in bytes of the redo log buffer. The default is 200m.

For large and lengthy transactions, increase LOG_BUFFER to reduce log file I/O

Verify that the redo log files are on separate and fast devices. Also verify that no other database files are causing contention with the redo log files. Query V$SYSTEM_EVENT for log files switch completion waits, which indicate the time waited for log switches.

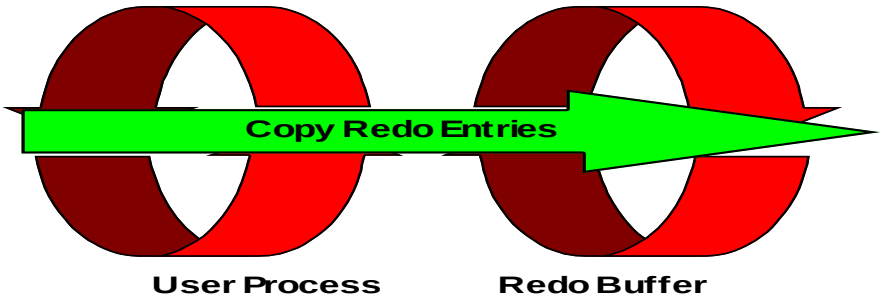
You can also query V$SESSION_WAIT to determine if any sessions are waiting for log buffer space.

```
Select SID, EVENT, SECONDS_IN_WAIT, STATE
From V$SESSION_WAIT
Where EVENT = 'log buffer space'
```

SID	EVENT	SECONDS_IN_WAIT	STATE
18	log buffer space	120	WAITING
41	log buffer space	150	WAITING

Our tuning goal is to have NO log buffer space waits. Waiting sessions indicate that the server process is waiting for space in the redo log buffer, which tells us that server processes are writing redo entries into the log buffer faster than LGWR can write to the redo log file. The value of 120 above indicates how long the session waited for this event. Again, consider increasing the size of the redo log buffer or move redo log files to a faster device.

REDO LOG BUFFERS UTILIZING LATCHES



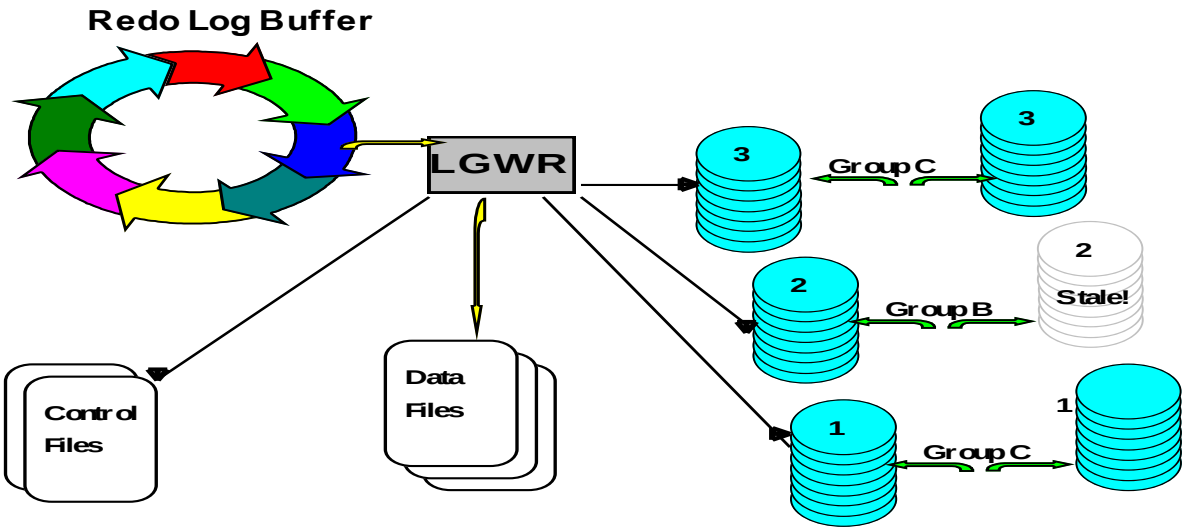
While the server process is working for the user process and space has been allocated, the server process copies the redo entries from the memory of its user process to the redo buffer in contiguous, sequential space.

Reducing REDO operations is a primary concern to assist in performance improvements. There are a few ways to decrease redo log activity during bulk load operations. Using the direct path option with SQL*Loader and using NOLOGGIN can significantly reduce redo log entries.

When you use SQL*Loader to bulk load data into the database, you have a choice between conventional and direct path loads. Conventional path loads cause redo log entries. With direct path loads, redo log entries are NOT generated if the database is in noarchivelog mode. If the database is in archivelog and you're using direct path, you can use the UNRECOVERABLE clause in the loader control file or use the NOLOGGING option on the database table to avoid logging redo.

If the NOLOGGING attribute is set for a table or index, only certain SQL statement will be affected. Use the NOLOGGING clause with CREATE TABLE, CREATE TABLE …AS SELECT, CREATE INDEX, and ALTER INDEX … REBUILD to significantly reduce redo log activity and improve performance of the statement.

LOG WRITER (LGWR)

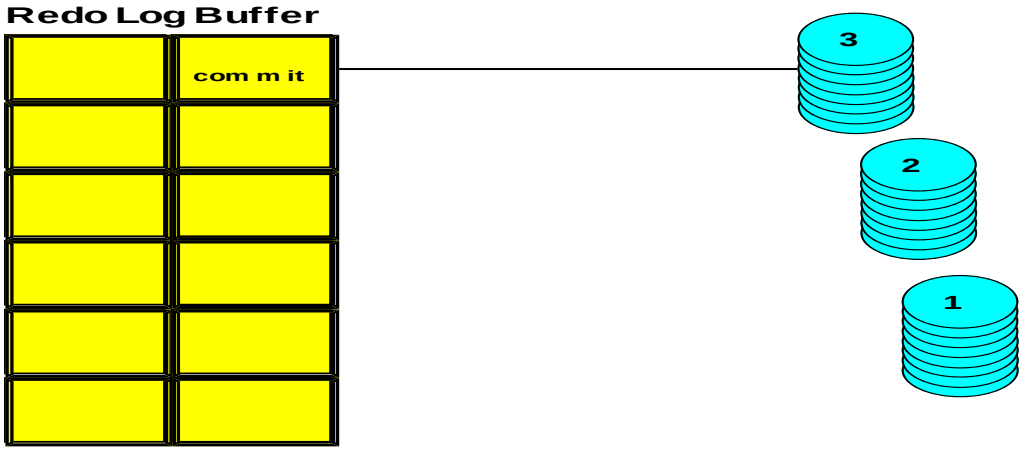


The Log Writer (LGWR) is responsible for writing all redo log entries that have been created since the last LGWR write from the redo log buffer to an on-line redo file, and for writing checkpoints in every data file and control file header when CKPT is not running. LGWR's sole responsibility is to manage the redo Log Buffer.

LGWR is signaled to write or flush the redo entries in the redo log buffer to an on-line redo log file:

- When the user process commits a transaction. In this case, LGWR writes a commit record to the log file
- Every 3 seconds
- When the redo log buffer is 1/3 full
- When DBWR writes modified buffers
- If the redo log file fills or another transaction commits before LGWR completes its current write

MORE ABOUT LOG WRITER



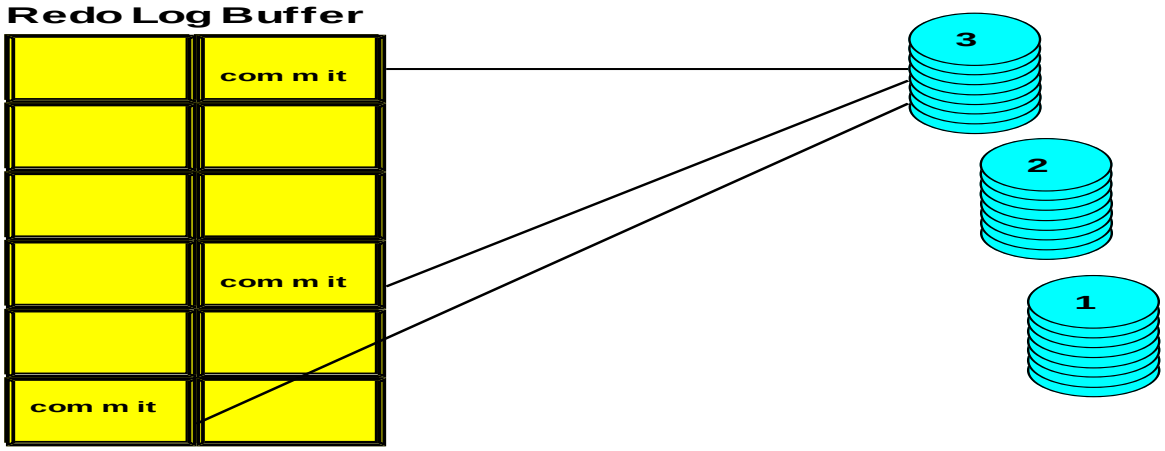
Once LGWR write entries from the redo log buffer, other server process can copy new entries into the buffer.

LGWR generally writes fast enough to always ensure space

LGWR writes redo log entries for committed and uncommitted transactions. the uncommitted transaction entries become permanent on if the transaction is committed. When they are committed it is called a fast commit. Uncommitted transaction are written if the redo buffer fills or if the redo entries are in the same buffer as entries from a committed transaction.

As soon as a user issues a commit, ORACLE immediately places the commit record in the redo log buffer. However, the corresponding changes in the database buffer cache is deferred until it is efficient to write them, i.e. before or after the commit.

LOG WRITE (LGWR) CONTINUED



A transaction is committed only when the commit redo entry record is written to the log file.

When a transaction is committed, a system change number (SCN) is assigned to the committed transaction so that recovery operations are synchronized. The SCN is recorded with all the corresponding redo entries for the transaction.

LGWR may write using group commits when activity is high.

If LGWR cannot write to a log file because a checkpoint has not completed, database operations are suspended until the checkpoint completes and the redo file is made available.

CHECKPOINTS DONE AT SHUTDOWN

We need to shutdown ORACLE

During a NORMAL or IMMEDIATE shutdown a database checkpoint is done. This checkpoint interrupts any checkpoint in progress.

Forced checkpoints issued by the database administrator interrupts any checkpoint in progress

You can monitor checkpoint activity by setting an initialization parameter and viewing the alert log, and also by querying the V$SYSTEM_EVENT view. By setting the value of LOG_CHECKPOINTS_TO_ALERT to TRUE, dynamically or in the spfile file. Oracle will create an alert log entry for each checkpoint. You then compare timestamps for each checkpoint to determine checkpoint frequency.

Another way to check on the frequency of log switches is to look at V$SYSSTAT.

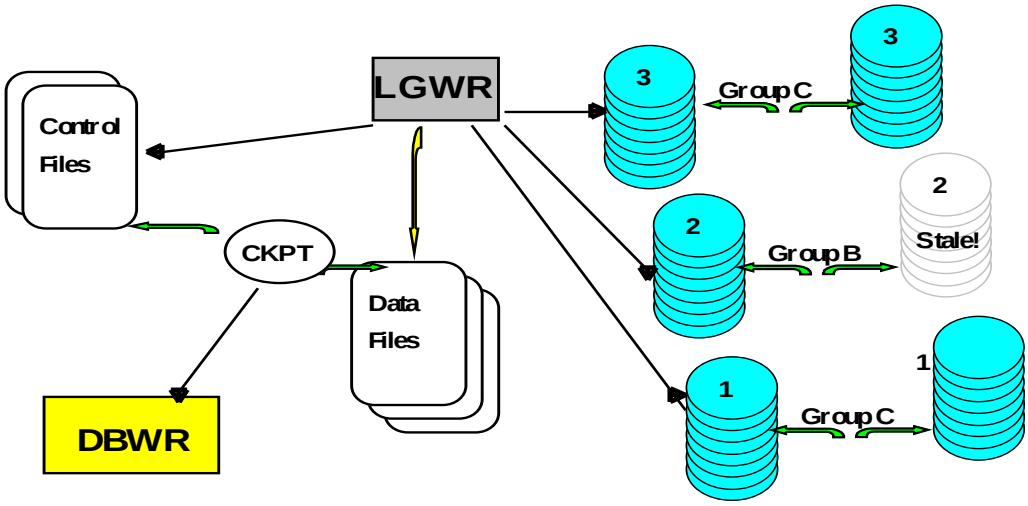
```
SELECT NAME, VALUE
From V$SYSSTAT
Where NAME LIKE 'background check%'
```

NAME	VALUE
background checkpoints started	7
background checkpoints completed	6

If the value of started is greater than the value of completed by more than 1, then checkpoints are not completing between log file switches, therefore, increase the size of the redo logs.

NOTE: To ensure that checkpoints occur only at log switches set LOG_CHECKPOINT_TIMEOUT=0. To force additional checkpoints set LOG_CHECKPOINT_TIMEOUT to a value less than the average time it takes LGWR to fill a redo log file. Look at the LGWR trace file for log switch times.

CHECKPOINT PROCESSING



Fast-Start Checkpointing allows the DBA to:

- Reduce the amount of time required for instance recovery
- specify number of seconds for MTTR

Instance recovery is improved by reducing amount of time for the instance recovery by limiting the number of dirty buffers and the lag between the checkpoint and the end of the redo log. Older dirty blocks are written first to ensure that the checkpoint moves forward and stays close behind the end of the redo log.

The initialization parameter FAST_START_MTTR_TARGET (MTTR) specifies the number of seconds for the expected mean time go recover the instance following a crash. Oracle will then vary the checkpointing processes to keep up with the MTTR. The maximum value for this parameter is 3,600 or 1 hour.

The FAST_START_IO_TARGET is similar to the MTTR setting except that this specifies the upper bound on the number of dirty buffers. This parameter is no longer used for oracle as well as LOG_CHECKPOINT_INTERVAL and LOG_CHECKPOINT_TIMEOUT when FAST_START_MTTR_TARGET is used.

The V$INSTANCE_RECOVERY view contains information about expected MTTR, actual and target redo blocks for recovery and estimated I/O's for recovery. Look at this view to determine ;your settings for FAST_START_MTTR_TARGET.

BLOCKING SESSION

User hold-ups

Possible Causes

When an extent in a rollback segment is full, the Oracle server attempts to reuse the next extent in the segment. If this new extent contains one active entry – that is, an entry for a transaction that is still active – it cannot be used. In these cases, a rollback segment allocates an additional extent. The transaction cannot skip an extent in the ring and continue writing to a subsequent extent. A transaction that has made only a few changes, but has been idle for a long time, could cause rollback segments to grow even though there are many free extents. In this situation, a log of free space is wasted and a database administrator may need to intervene to avoid excessive rollback segment growth.

Solution

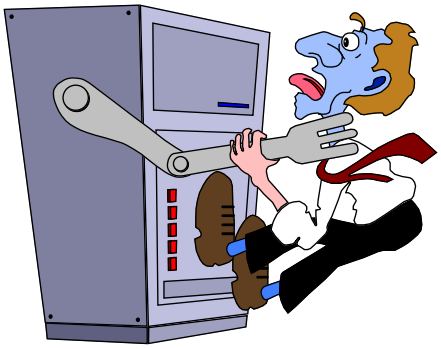
Query the V$ROLLSTAT, V$SESSION, and V$TRANSACTION view to find any blocking transaction.

```
SQL> SELECT s.sid, s.serial#, t.start_time, t.xidusn, s.username
     FROM v$session s, v$transaction t, v$rollstat r
     WHERE s.saddr = t.ses_addr
     AND       t.xidusn = r.usn
     AND       ((r.curext = t.start_uext-1) or
     ((r.curext = r.extents-1) and t.start_uext=0));
```

SID	SERIAL#	START_TIME	XIDUSN	USERNAME
11	23	02/18/2002 08:30:32	2	SYSTEM

Check to see if the transaction can be ended by the user. If not, it may be necessary to kill the session.

ERROR IN TAKING A TABLESPACE OFFLINE



You did WHAT????

Problem Diagnosis and Resolutions

If a tablespace contains one or more active rollback segments, it cannot be taken offline. The session executing the statement will receive an ORA-01546 error.

Solution

Perform the following steps:

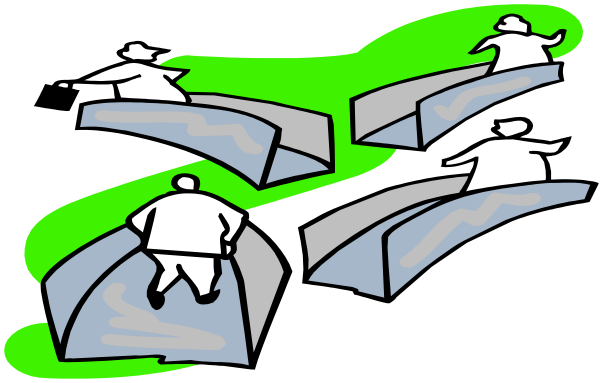
1. Check V$TRANSACTION to find which transactions are currently using these undoegments.

2. Use V$SESSION to obtain the username and session information.

3. Kill the session or have the user end the transaction

4. Take the tablespace offline

AUTOMATIC UNDO MANAGEMENT



The rollback segments managed without intervention

Automatic undo management completely automates undo data management. An instance running in automatic undo management mode creates and manages undo segments without the DBA intervening. In AUTO mode, the DBA doesn't need to specify the number, size and location of the undo segments. Oracle recommends using automatic undo management instead of manual.

Automatic UNDO MANAGEMENT INITIALIZATION PARAMETERS

- UNDO_MANAGEMENT Specifies which undo space management mode the system will use. When set to AUTO, the instance starts in automatic undo management mode. When set to MANUAL, the instance starts in manual undo management mode. This is not a dynamic parameter

- UNDO_TABLESPACE Names the undo tablespace that will be used when the instance starts up. This parameter cannot be used if the instance is in MANUAL undo management mode.

- UNDO *SUPRESS*_ERRORS Setting this initialization parameter to TRUE suppresses error messages caused by references to manually configured rollback segments when the instance is running in automatic undo management mode. This happens when the SET TRANSACTION clause is used.

- UNDO_RETENTIOIN This dynamically configurable parameter specifies in seconds how long to retain committed undo information in the undo segment. The default value is 900 seconds (15 minutes). Queries that require older information require the DBA to set this to a larger value. Keep in mind that the size of the UNDO_TABLESPACE will grow while these transactions are running.

CREATING AND MAINTAINING THE UNDO TABLESPACE

If UNDO_MANAGEMENT is set to TRUE in the spfile file when the Oracle9i database is created, a default undo tablespace IS created called SYS_UNDOTBS (depending on your Oracle release and OS) with a datafile name of $ORACLE_HOME/dbs/dbu1<SID>.dbf with AUTOEXTEND ON. If the parameter UNDO_TABLESPACE is used and it does not match the name of the tablespace used in the CREATE DATABASE command, Oracle generates an error.

DBA's can also create an UNDO TABLESPACE with the CREATE UNDO TA BLESPACE command after the database is created. Then change the spfile parameter to reflect the tablespace being used for UNDO by again using the UNDO_TABLESPACE parameter.

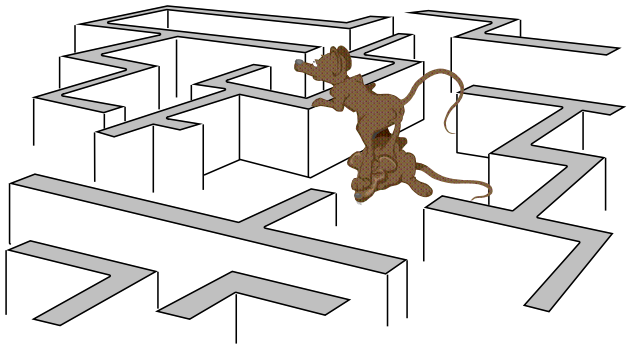
Some restrictions on creating an UNDO TABLESPACE:

1. You cannot create database objects in this tablespace.
2. You can specify the extent_management_clause and datafile name only when creating the tablespace.

You can monitor UNDO acitivity and the amount of space used by viewing V$UNDOSTAT. Each row in the represents undo statistics for a ten-minute interval. This view can also be used to identify how much space is required for the UNDO_TABLESPACE. The following query can be used to calculate the undo space required as part of viewing the undo activity per second, the block size, and the retention requirement. The value returned indicates the megabytes required.

```
Select (UR * (UPS * OVERHEAD) + OVERHEAD) as "Bytes"
From (Select value as UR from V$PARAMETER where NAME = 'undo_retention),
      (Select (sum(undoblks) / sum(((END_TIME – BEGIN_TIME) * 86400))) as UPS
from V$UNDOSTAT),  (SELECT value as OVERHEAD from V$PARAMETER where
                          NAME = 'db_block_size')
```

Performance Hints on UNIX-Based Systems



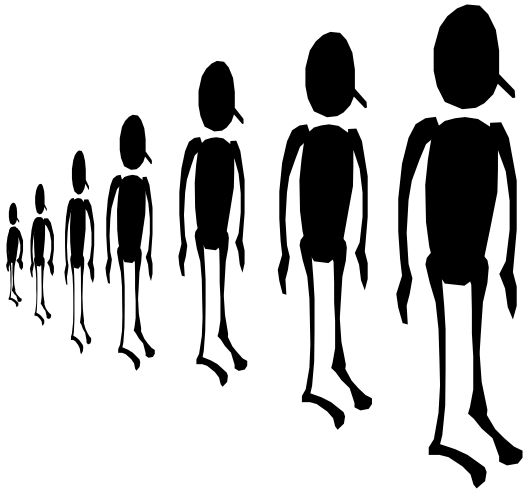
On UNIX systems, try to establish a good ratio between the amount of time the operating system spends fulfilling system calls and doing process scheduling and the amount of time the application runs. The goal should be to run 60% to 75% of the time in application mode (often called user mode) and 25% to 40% of the time in operating system mode. If you find that the system is spending 50% of its time in each mode, then determine what is wrong.

The ratio of time spent in each mode is only a symptom of the underlying problem, which might involve the following:

- Paging or swapping
- Executing too many operating system calls
- Running too many processes

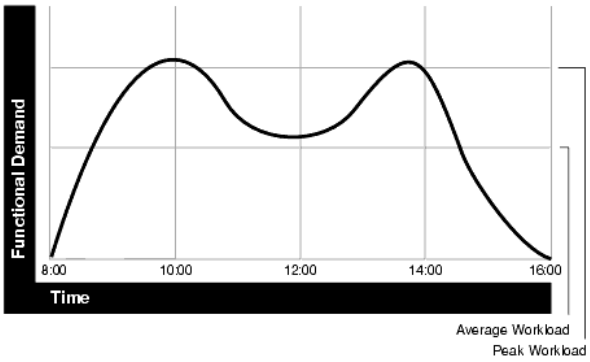
If such conditions exist, then there is less time available for the application to run. The more time you can release from the operating system side, the more transactions an application can perform.

WORKLOAD LEVELS ON A CPU



Workload is an important factor when evaluating your system's level of CPU utilization. During peak workload hours, 90% CPU utilization with 10% idle and waiting time can be acceptable. Even 30% utilization at a time of low workload can be understandable. However, if your system shows high utilization at normal workload, then there is no room for a peak workload. For example, Figure 16-1 illustrates workload over time for an application having peak periods at 10:00 AM and 2:00 PM.

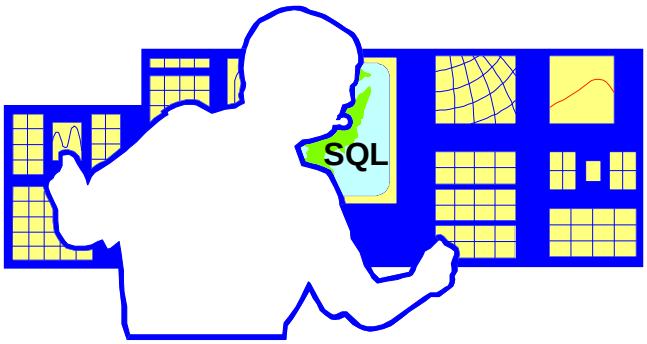
Figure 16-1 Average Workload and Peak Workload



This example application has 100 users working 8 hours a day. Each user entering one transaction every 5 minutes translates into 9,600 transactions daily. Over an 8-hour period, the system must support 1,200 transactions an hour, which is an average of 20 transactions a minute. If the demand rate were constant, then you could build a system to meet this average workload.

However, usage patterns are not constant and in this context, 20 transactions a minute can be understood as merely a minimum requirement. If the peak rate you need to achieve is 120 transactions a minute, then you must configure a system that can support this peak workload.

Finding System CPU Utilization



Oracle statistics report CPU use by Oracle sessions only, whereas every process running on your system affects the available CPU resources. Therefore, tuning non-Oracle factors can also improve Oracle performance.

Use operating system monitoring tools to determine what processes are running on the system as a whole. If the system is too heavily loaded, check the memory, I/O, and process management areas described later in this section.

Tools such as `sar -u` on many UNIX-based systems let you examine the level of CPU utilization on your entire system. CPU utilization in UNIX is described in statistics that show user time, system time, idle time, and time waiting for I/O. A CPU problem exists if idle time and time waiting for I/O are both close to zero (less than 5%) at a normal or low workload.

On NT, use Performance Monitor to examine CPU utilization. Performance Manager provides statistics on processor time, user time, privileged time, interrupt time, and DPC time. (NT Performance Monitor is not the same as Performance Manager, which is an Oracle Enterprise Manager tool.)

Paging and Swapping

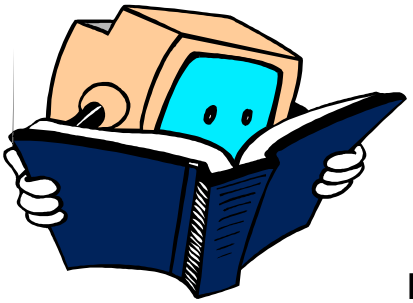
Use tools such as `sar` or `vmstat` on UNIX or Performance Monitor on NT to investigate the cause of paging and swapping.

Oversize Page Tables

On UNIX, if the processing space becomes too large, then it can result in the page tables becoming too large. This is not an issue on NT.

Checking I/O Management

Thrashing is an I/O management issue. Ensure that your workload fits into memory, so the machine is not **thrashing** (swapping and paging processes in and out of memory). The operating system allocates fixed portions of time during which CPU resources are available to your process. If the process wastes a large portion of each time period checking to be sure that it can run and ensuring that all necessary components are in the machine, then the process might be using only 50% of the time allotted to actually perform work.



LAB 1 DATABASE TUNING

1. THE SGA IS COMPOSED OF WHAT THREE PRIMARY UNITS?

   A. _____

   B. _____

   C. _____

2. What command(s) can you use to examine the SGA memory structure?

_____

3. What three parameters in the spfile control the size of the SGA and what are the current defaults for each?

   1. _____

   2. _____.

   3. _____

4. The database buffer cache tracks what kind of buffers and activity?

   _____

.

5. What dynamic performance table allows you to monitor the database buffer cache? What query can you use to identify the ratio of datablocks currently in the buffer cache to physical reads to place data in the buffers from disk? What percentage of database hits in the buffer constitute good performance?

   _____

   _____

6. The SGA shared pool is composed primarily of two entities. Name those two entities

   A. _____

   B. _____

7. To ensure that the SGA is performing properly what script would you use to identify performance problems with the SHARED SQL AREA? What percentage of misses would require an increase in the SGA? What are the current “hit” ratios in the database?

   _____

   _____

8. The Data Dictionary should be entirely in memory if possible. What script identifies the hits and misses for the Data Dictionary? What percentage of misses would require an increase in the SGA? What are the current “hit” ratios in the database?

   _____

   _____

   _____

9. What is the purpose of the UNDO Segments?