IBM Oracle Technical Brief

Oracle Architecture and Tuning on AIX

Damir Rubic

IBM SAP & Oracle Solutions Advanced Technical Support

Version: 1.2
Date: 08 July 2008

Acknowledgements

Thank you to Dale Martin, Jim Dilley, Ralf Schmidt-Dannert for all their help regarding this and many other Oracle & IBM Power Systems performance related subjects.

- ***Disclaimers***

IBM has not formally reviewed this paper. While effort has been made to verify the information, this paper may contain errors. IBM makes no warranties or representations with respect to the content hereof and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose. IBM assumes no responsibility for any errors that may appear in this document. The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

- ***Copyrights***

- ***Feedback***

Please send comments or suggestions for changes to drubic@us.ibm.com.

- ***Version Updates***

  - Version 1.0 - initial version
  - Version 1.1 - added architectural and tuning elements related to AIX 5.3
  - Version 1.2 - added architectural and tuning elements related to AIX 6.1

Introduction

This paper is intended for IBM Power Systems customers, IBM technical sales specialists, and consultants who are interested in learning more the requirements involved in building and tuning an Oracle RDBMS system for optimal performance on the AIX platform.

This white paper contains best practices which have been collected during the extensive period of time my team colleagues and I have spent working in the Oracle RDBMS based environment. It is primarily focused on the AIX versions 5L & 6.1 and Oracle 9i & 10g.

This paper begins with a short description of the most important Oracle DB architectural elements. It continues with an overview of the AIX-related tuning elements that are most crucial for optimal DB activity.

This document can be expanded into many different OS or DB-related directions. Additional information on related topics is included in Appendix A of this paper, as well as references to supporting documentation. However, this paper is not focused on the application tuning area. Application performance tuning is a subject too broad to be covered in a white paper of this length.

Additionally, an equally important prerequisite for optimal DB activity is the careful planning of the DB/Storage layout. More information on DB/Storage layout can be found in the Appendix A.

1. Oracle Database Architecture

1.1. Database Structure

An **Oracle database** is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

The database has **logical structures** and **physical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

1.1.1. Logical Structure

An Oracle database is made up of several logical storage structures, including *data blocks*, *extents* and *segments*, *tablespaces*, and *schema objects*.

The actual physical storage space in the datafiles is logically allocated and deallocated in the form of Oracle data blocks. Data blocks are the smallest unit of I/O in an Oracle database. Oracle reserves a portion of each block for maintaining information, such as the address of all the rows contained in the block and the type of information stored in the block.

An extent is a collection of contiguous data blocks. A table is comprised of one or more extents. The very first extent of a table is known as the *initial extent*. When the data blocks of the initial extent become full, Oracle allocates an *incremental extent*. The incremental extent does not have to be the same size (in bytes) as the initial extent.

A segment is the collection of extents that contain all of the data for a particular logical storage structure in a tablespace, such as a table or index. There are four different types of segments, each corresponding to a specific logical storage structure type:

- Data segments
- Index segments
- Rollback segments
- Temporary segments

Data segments store all the data contained in a table, partition, or cluster. Likewise, index segments store all the data contained in an index. For backward compatibility, rollback segments are used to hold the previous contents of an Oracle data block prior to any change made by a particular transaction. If any part

of the transaction should not complete successfully, the information contained in the rollback segments is used to restore the data to its previous state. In Oracle9i and 10g, this is achieved using *automatic undo* and *undo tablespaces*, which allows better control and use of the server resources.

Rollback segments are also used to provide *read-consistency*. There are two different types of read-consistency: *statement-level* and *transaction-level*.

Statement-level read consistency ensures that all of the data returned by an individual query comes from a specific point in time: the point at which the query started. This guarantees that the query does not see changes to the data made by other transactions that have committed since the query began. This is the default level of read-consistency provided by Oracle.

In addition, Oracle offers the option of enforcing transaction-level read consistency. Transaction-level read consistency ensures that all queries made within the same transaction do not see changes made by queries outside of that transaction but can see changes made within the transaction itself. These are known as *serializable* transactions.

Temporary segments are used as temporary workspaces during intermediate stages of a query's execution. They are typically used for sort operations that cannot be performed in memory. The following types of queries may require a temporary segment:

- SELECT.....ORDER BY
- SELECT.....GROUP BY
- SELECT.....UNION
- SELECT.....INTERSECT
- SELECT.....MINUS
- SELECT DISTINCT.....
- CREATE INDEX....

Tablespaces group related logical entities or objects together in order to simplify physical management of the database. Tablespaces are the primary means of allocating and distributing database data at the physical disk level. Tablespaces are used to:

- Control the physical disk space allocation for the database
- Control the availability of the data by taking the tablespaces online or off-line
- Distribute database objects across different physical storage devices to improve performance
- Regulate space for individual database users

Every Oracle database contains at least one tablespace named SYSTEM. The SYSTEM tablespace contains the data dictionary tables for the database used to describe its structure. Schema objects are the logical structures used to refer to the database's data. A few examples of schema objects would be tables, indexes, views, and stored procedures. Schema objects, and the relationships between them, constitute the relational design of a database.

1.1.2. Physical storage structures

An Oracle database is made up of three different types of physical database files: *datafiles*, *redo logs*, and *control files*.

An Oracle database must have one or more datafiles in order to operate. Datafiles contain the actual database data logically represented in the form of tables or indexes. At the operating system level, datafiles can be implemented in a several different ways (JFS, JFS2, GPFS, Veritas FS, Oracle ASM etc). In this document we will focus only on the JFS or JFS2 based files and raw devices. The data contained in the datafiles is read from disk into the memory regions.

An Oracle tablespace is comprised of one or more datafiles. A datafile cannot be associated with more than one tablespace, nor can it be used by more than one database. At creation time, the physical disk space associated with a datafile is pre-formatted, but does not contain any user data. As data is loaded into the system, Oracle reserves space for data or indexes in the datafile in the form of extents.

Redo logs are used by Oracle to record all changes made to the database. Every Oracle database must have at least two redo logs in order to function. The redo log files are written to in a circular fashion; when the current online log fills up, Oracle begins writing to the next available online redo log. In the event of a failure, changes to the Oracle database can be reconstructed using the information contained in the redo logs. Due to their importance, Oracle provides a facility for mirroring or *multiplexing* the redo logs so that two (or more) copies of the log are available on disk.

The control file describes the physical structure of the database. It contains information, such as the database name, date, and time the database was created, and the names and locations of all the database data files and redo logs. Like the redo logs, Oracle can have multiple copies of the control file to protect against logical or physical corruption.

1.2. Instance and Application Processes

A process is defined as a *thread of control* used in an operating system to execute a particular task or series of tasks. Oracle utilizes three different types of processes to accomplish these tasks:

- **User or client processes**
- **Server processes**
- **Background processes**

User processes are created to execute the code of a client application program. The user process is responsible for managing the communication with the Oracle server process using a *session*. A session is a specific connection of a user application program to an Oracle instance. The session lasts from the time that the user or application connects to the database until the time the user disconnects from the database.

Server processes are created by Oracle to service requests from connected user processes. They are responsible for interfacing with the database to carry out the requests of user processes. The number of user processes per server process is dependent on the configuration of Oracle. In a *dedicated server* configuration, one server process is spawned for each connected user process. In a *multi-threaded server* configuration, user processes are distributed among a pre-defined number of server processes.
Oracle background processes are created upon database startup or initialization. Some background processes are necessary for normal operation of the system, while others are only used to perform certain database maintenance or recovery related functions.

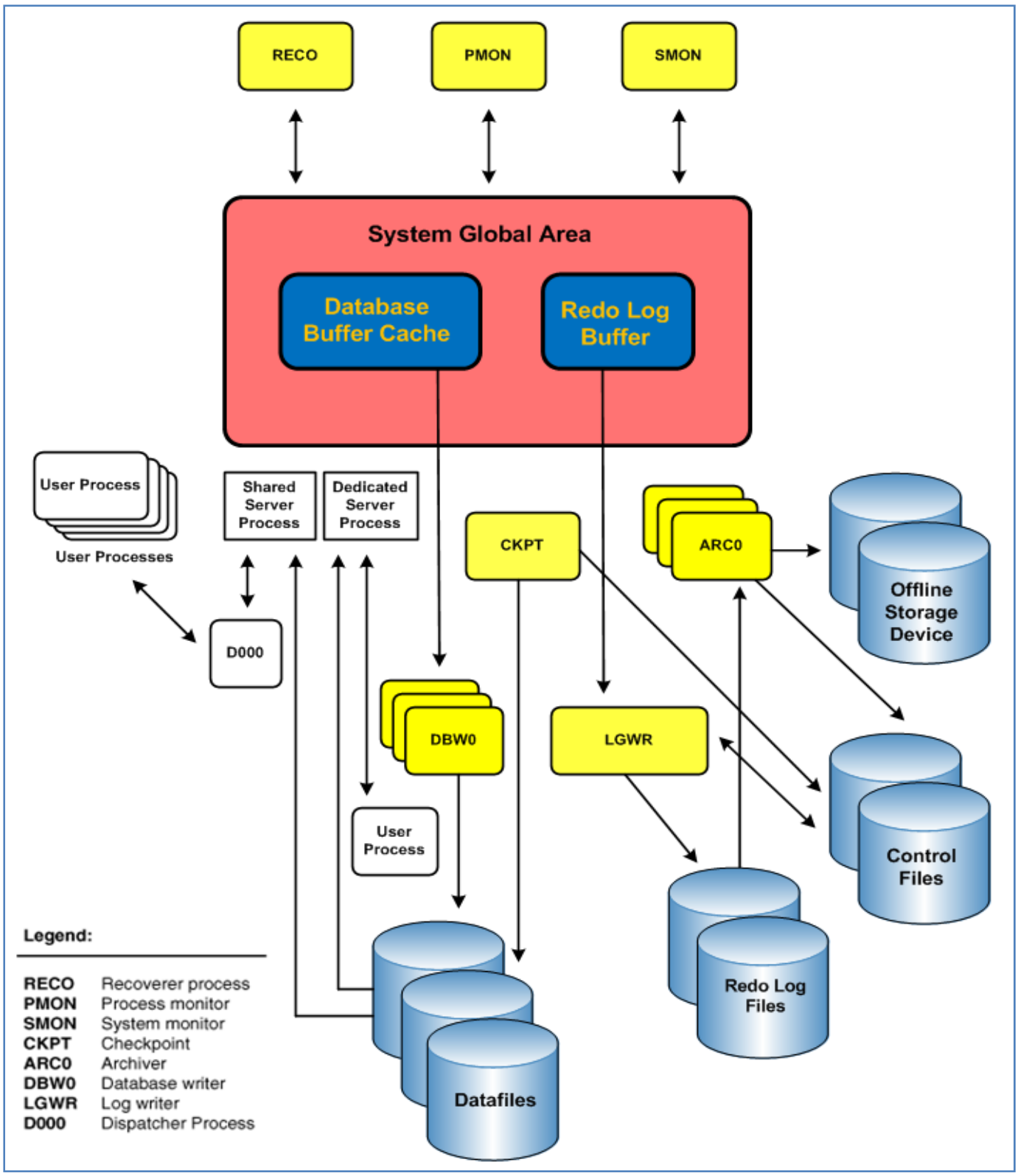


Figure 1-1 Oracle Process Architecture

The Oracle background processes include:

- **Database Writer (DBWn)**

The database writer process is responsible for writing modified or *dirty* database buffers from the database buffer cache to disk. It uses a least recently used (LRU) algorithm to ensure that the user processes always find free buffers in the database buffer cache. Dirty buffers are written to disk using a single block writes. Additional database writer processes can be configured to improve write performance if necessary. On AIX, enabling asynchronous I/O eliminates the need for multiple database writer processes and should yield better performance.

- **Log Writer (LGWR)**

The log writer process is responsible for writing modified entries from the redo log buffer to the online redo log files on disk. This occurs when one of the following conditions is met:

  - Three seconds have elapsed since the last buffer write to disk.
  - The redo log buffer is one-third full.
  - The DBWn process needs to write modified data blocks to disk for which the corresponding redo log buffer entries have not yet been written to disk.
  - A transaction commits.

A *commit record* is placed in the redo log buffer when a user issues a COMMIT statement, at which point the buffer is immediately written to disk. The redo log buffer entries are written in First-In-First-Out (FIFO) order. This means that once the commit record has been written to disk, all of the other log records associated with that transaction have also been written to disk. The actual modified database data blocks are written to disk at a later time, a technique known as *fast commit*. A System Change Number (SCN) defines a committed version of a database at a point in time. Each committed transaction is assigned a unique SCN.

- **Checkpoint Process (CKPT)**

The checkpoint process is responsible for notifying the DBWn process that the modified database blocks in the SGA need to be written to the physical datafiles. It is also responsible for updating the headers of all Oracle datafiles and the controlfile(s) to record the occurrence of the most recent checkpoint.

- **Archiver (ARCn)**

The archiver process is responsible for copying the online redo log files to an alternate physical storage location once they become full. The ARCH process exists only when the database is configured for ARCHIVELOG mode, and automatic archiving is enabled.

- **System Monitor (SMON)**

The system monitor process is responsible for performing recovery of the Oracle instance upon startup. It is also responsible for performing various other administrative functions, such as cleaning up temporary segments that are no longer in use and coalescing free extents in dictionary managed tablespaces.

- **Process Monitor (PMON)**

The process monitor is responsible for cleaning up after failed user processes. This includes such tasks as removing entries from the process list, releasing locks, and freeing up used blocks in the database buffer cache associated with the failed process. It is also responsible for starting dispatcher or server processes that have unexpectedly stopped.

- **Recover (RECO)**

The recover process is responsible for recovering all in-doubt transactions that were initiated in a distributed database environment. RECO contacts all other databases involved in the transaction to remove any references associated with that particular transaction from the pending transaction table. The RECO process is not present at instance startup unless the DISTRIBUTED_TRANSACTIONS parameter is set to a value greater than zero and distributed transactions are allowed.

- **Job coordinator (CJQn)**

The job coordinator process is the scheduler in an Oracle instance. It is responsible for starting jobs processes that will execute batch processing. These jobs are PL/SQL statements or procedure in the instance. When a job has executed, CJQn will spawn a job queue slave process named J000 to J999. Thus, slave process then will proceed to execute job processing. If the JOB_QUEUE_PROCESSES is set to 0, the job coordinator will not start.

- **Queue Monitor Processes (QMNn)**

The queue monitor process is an optional background process for Oracle Streams Advanced Queuing, which monitors the message queues. You can configure up to 10 queue monitor processes. These processes, like the job queue processes, are different from other Oracle background processes in that process failure does not cause the instance to fail.

- **Dispatcher (Dnnn)**

Dispatcher processes are only present in a multi-threaded server configuration. They are used to allow multiple user processes to share one or more server processes. A client connection request is received by a network listener process, which, in turn, passes the request to an available dispatcher process, which then routes the request to an available server process. If no dispatcher processes are available, the listener process starts a new dedicated server process and connects the user process directly to it.

There are several other background processes that might be running. These can include the following:

- **MMON**

Performs various manageability-related background tasks, for example:

  - Issuing alerts whenever a given metrics violates its threshold value
  - Taking snapshots by spawning additional process (MMON slaves)
  - Capturing statistics value for SQL objects which have been recently modified

- **MMNL**

Performs frequent and light-weight manageability-related tasks, such as session history capture and metrics computation.

- **MMAN**

Is used for internal database tasks.

- **RBAL**

Coordinates rebalance activity for disk groups in an Automatic Storage Management instance. It performs a global open on Automatic Storage Management disks. ORBn performs the actual rebalance data extent movements in an Automatic Storage Management instance. There can be many of these at a time, called ORB0, ORB1, and so forth.

- **OSMB**

Is present in a database instance using an Automatic Storage Management disk group. It communicates with the Automatic Storage Management instance.

The Oracle RAC Specific Instance Processes include:

- **Global Cache Service Processes (LMSn)**

LMSn processes control the flow of messages to remote instances and manage global data block access. LMS*n* processes also transmit block images between the buffer caches of different instances. This processing is part of the Cache Fusion feature. *n* ranges from 0 to 9 depending on the amount of messaging traffic.

- **Global Enqueue Service Monitor (LMON)**

Monitors global enqueues and resources across the cluster and performs global enqueue recovery operations. Enqueues are shared memory structures that serialize row updates.

- **Global Enqueue Service Daemon (LMD)**

Manages global enqueue and global resource access. Within each instance, the LMD process manages incoming remote resource requests.

- **Lock Process (LCK)**

Manages non-Cache Fusion resource requests such as library and row cache requests.

- **Diagnosability Daemon (DIAG)**

Captures diagnostic data about process failures within instances. The operation of this daemon is automated and it updates an alert log file to record the activity that it performs.

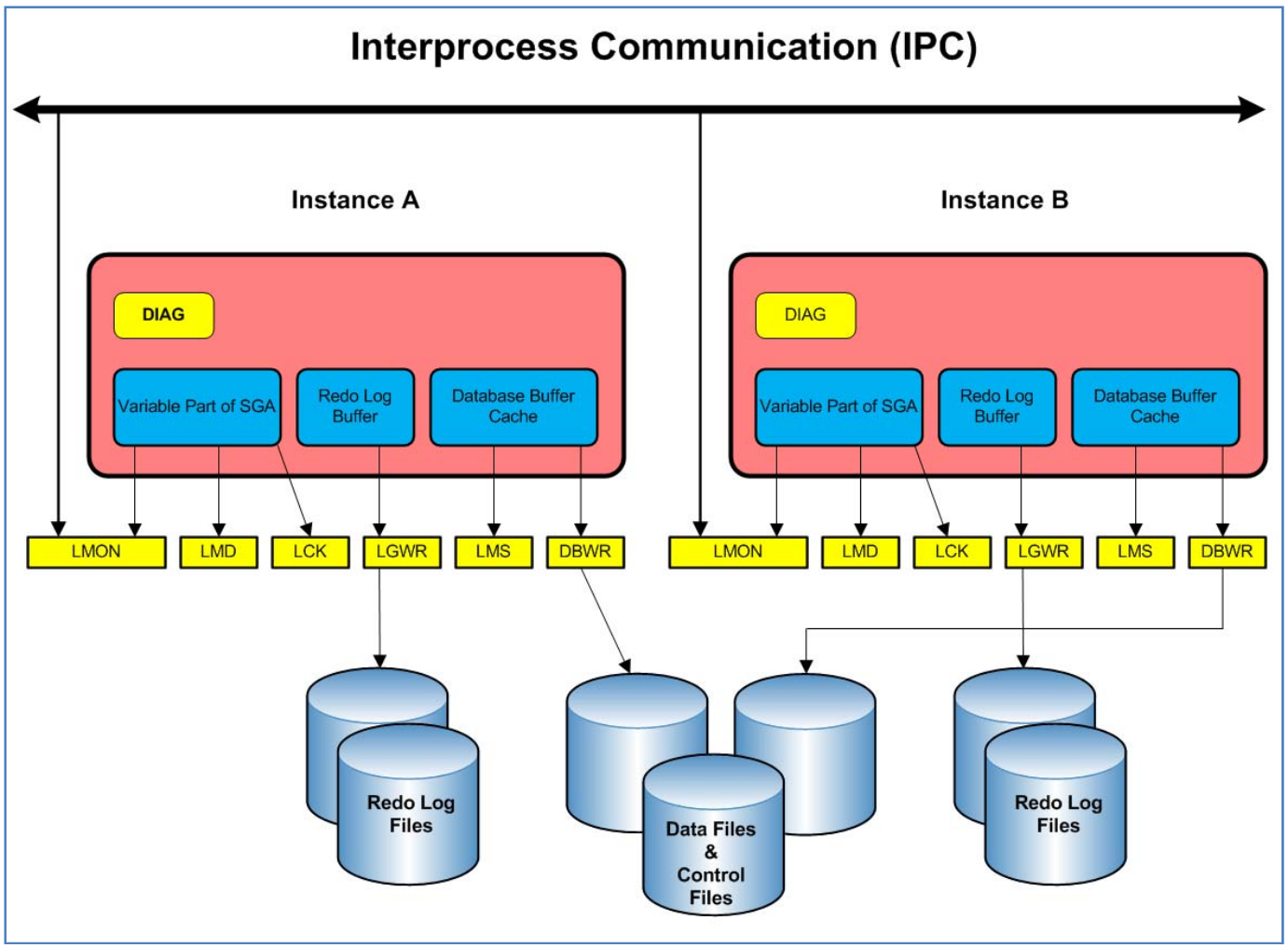


Figure 1-2 Oracle RAC Specific Instance Processes

1.3. Oracle memory Structures

Oracle utilizes several different types of memory structures for storing and retrieving data in the system. These include the *System Global Area* (SGA) and *Program Global Areas* (PGA).

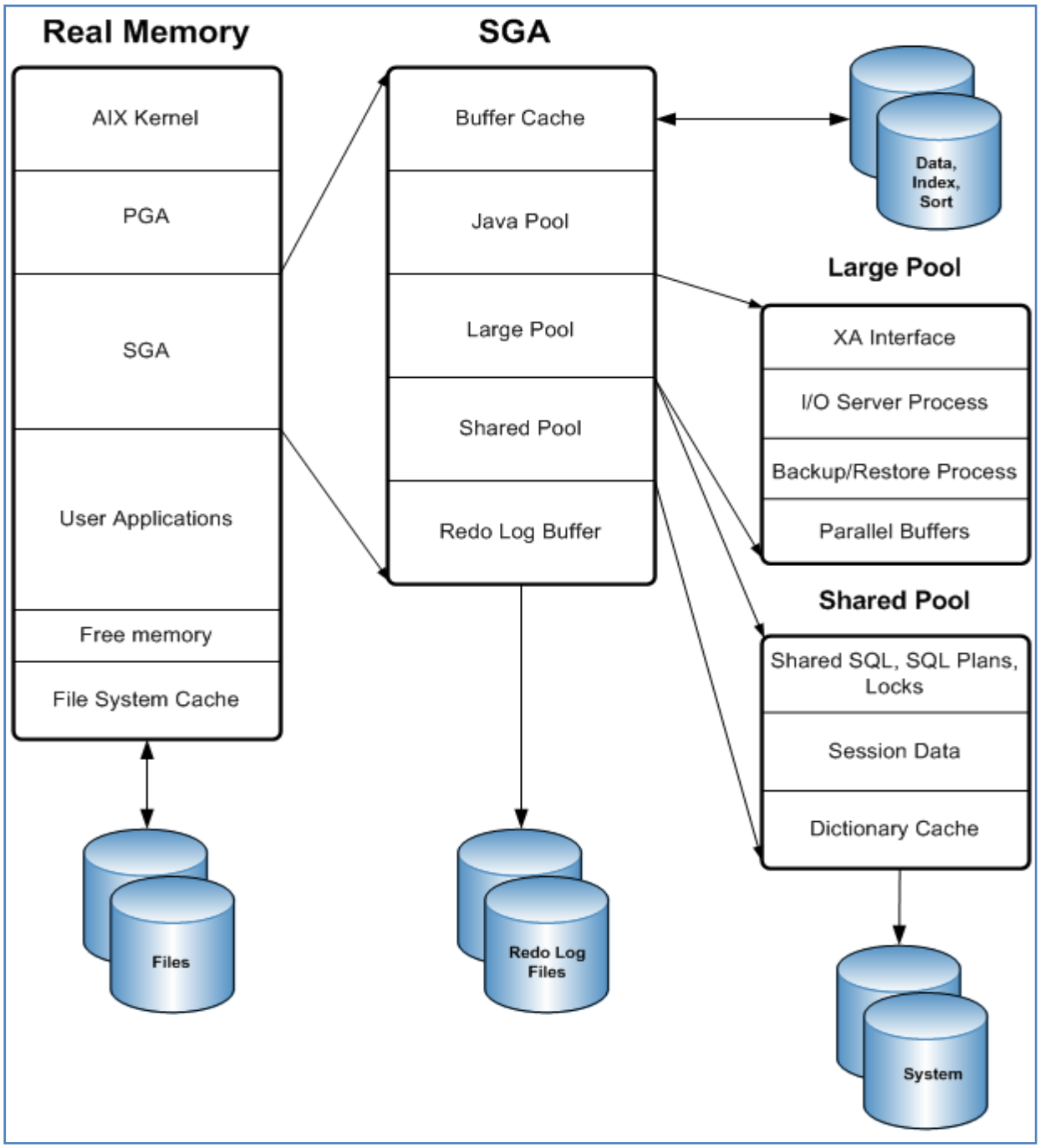


Figure 1-3 Oracle Memory structures

The Oracle SGA is a shared memory region used to hold data and internal control structures of the database. Shared memory region details can be displayed in AIX with the ipcs -ma command. The Oracle SGA and its associated background processes are known as an Oracle *instance*. The instance identity is called by the short name *SID*. This short name is used in all Oracle processes connected to this instance. To connect to a particular instance, set the shell variable $ORACLE_SID to the SID or specify it in the connect command at SQL*Plus. The SGA memory region is allocated upon instance startup and de-allocated when the instance is shut down and is unique to each database instance. The SGA is dynamically managed by Oracle, while the instance is up.
The information contained in the SGA is logically separated into three to five different areas: The *database buffer cache*, the *redo log buffer*, the *shared pool*, the *Java pool* (optional) and the *large pool (optional)*.

The database buffer cache consists of Oracle database data blocks or *buffers* that have been read from disk and placed into memory. These buffers are classified as either *free*, *dirty* or *pinned* and are organized in two lists: the *write list* and the *LRU list*. Free buffers are those that have not yet been modified and are available for use. Dirty buffers contain data that has been modified but has not yet been written to disk and are held in the write list. Lastly, pinned buffers are buffers that are currently being accessed.

Oracle uses a Least Recently Used (LRU) algorithm to age the dirty data blocks from memory to disk. A LRU list is used to keep track of all available buffers and their state (dirty, free, or pinned). Infrequently accessed data is moved to the end of the LRU list where it will eventually be written to disk by the Oracle DBWn process should an additional free buffer be requested but not available.

The size of the database buffer cache is determined by a combination of some Oracle initialization parameters. It's possible to set different block sizes for a database by specifying different cache sizes for each of these non-standard block sizes. The `DB_BLOCK_SIZE` parameter specifies the default or standard block size and the `DB_CACHE_SIZE` parameter sets the size of the cache using the blocks of `DB_BLOCK_SIZE`. To specify different block sizes, use the initialization parameter `DB_nK_CACHE_SIZE`, where n ranges from 2 KB to 32 KB. Also, the DBA can change these parameters while the instance is up and running, except for the `DB_BLOCK_SIZE` parameter, which requires the database to be recreated.

The Java pool is used to hold Java execution code and classes information if the Java option is turned on to a specific Oracle instance.

The redo log buffer is used to store information about changes made to data in the database. This information can be used to reapply or *redo* the changes made to the database should a database recovery become necessary. The entries in the redo log buffer are written to the online redo logs by the LGWR process. The size of the redo log buffer is determined by the `LOG_BUFFER` parameter in the Oracle initialization file.

The shared pool area stores memory structures, such as the shared SQL areas, private SQL areas, and the data dictionary cache and, if large pool has not been allocated, the buffers for parallel execution messages. Shared SQL areas contain the parse tree and execution plan for SQL statements. Identical SQL statements share execution plans. One memory region can be shared for multiple identical Data Manipulation

Language (DML) statements, thus saving memory. DML statements are SQL statements that are used to query and manipulate data stored in the database, such as SELECT, UPDATE, INSERT, and DELETE. Private SQL areas contain Oracle bind information and run-time buffers. The bind information contains the actual data values of user variables contained in the SQL query.

The data dictionary cache is used to hold information pertaining to the Oracle data dictionary. The Oracle data dictionary serves as a roadmap to the structure and layout of the database. The information contained in the data dictionary is used during Oracle's parsing of SQL statements.

The buffers for parallel execution hold information needed to synchronize all the parallel operations in the database. This is allocated from SGA only when the large pool is not configured.

The large pool is an optional memory area that can be used to address shared memory contention. It holds information such as the Oracle XA interface (the interface that Oracle uses to enable distributed transactions), I/O server processes, backup and restore operations, and if the initialization parameter `PARALLEL_AUTOMATIC_TUNING` is set to TRUE, the buffer for parallel execution. Large pool is enabled by setting the `LARGE_POOL_SIZE` to a number greater than 0.

Starting with Oracle 9i is a new feature called *Dynamic SGA* that allows optimized memory usage by instance. Also, it allows increasing or decreasing Oracle's use of physical memory, with no downtime, because the database administrator may issue ALTER SYSTEM commands to change the SGA size, while the instance is running. This feature is also available in Oracle 10g and 11g.

The Oracle PGA is the collection of non-shared memory regions that each contain data and control information for an individual server process. Whenever a server process is started, PGA memory is allocated for that process. The PGA memory can only be used by the particular server process it has been allocated for. In general, PGA contains a private SQL area (storing bind information and run-time structures that are associated with a shared SQL area) and a session memory that holds session specific variables, such as logon information. If Multi Threaded Server (MTS) architecture is being used, the server process and associated PGA memory may be "shared" by multiple client processes.

The PGA may also be used as a work area for complex queries making use of memory-intensive operators like the following ones:

- Sort operations, like ORDER BY, GROUP BY, ROLLUP, and WINDOW
- HASH JOINS
- BITMAP MERGE
- BITMAP CREATE

If the session connects to a dedicated Oracle server, PGA is automatically managed. The database administrator just needs to set the `PGA_AGGREGATE_TARGET` initialization parameter to the target amount of memory he/she wants Oracle to use for server processes. `PGA_AGGREGATE_TARGET` is a target value only. In some situations, the actual aggregate PGA usage may be significantly higher than the target value.

2. AIX Configuration & Tuning for Oracle

Disclaimer: The suggestions presented here are considered to be basic configuration "starting points" for general Oracle database workloads. Individual customer workloads will vary. Ongoing performance monitoring and tuning is recommended to ensure that the configuration is optimal for the customer's particular workload characteristics.

This section contains information on common AIX and system administration items related to Oracle database environments. The items in this section are necessary to ensure a good performing database system, and these are worth checking before engaging in fine, detailed-level database tuning. Additionally, for better understanding of this chapter, it would be good if one is familiar with the basics of the AIX VMM environment (commands and structures).

For that specific reason I have included several paragraphs that will hopefully clarify generic set of the AIX VMM elements.

2.1. Overview of the AIX Virtual Memory Manager (VMM)

The Virtual Memory Manager (VMM) services memory requests from the system and its applications. Virtual-memory segments are partitioned in units called pages; each page is either located in real physical memory (RAM), 'paged out' to disk to make room for other pages or stored on disk until it is needed. AIX uses virtual memory to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the VMM.

The virtual address space is partitioned into segments. A segment is a 256 MB, contiguous portion of the virtual-memory address space into which a data object can be mapped. Process addressability to data is managed at the segment (or object) level so that a segment can be shared between processes or maintained as private. For example, processes can share code segments yet have separate and private data segments.

2.1.1. Real-Memory Management

Virtual-memory segments are partitioned into fixed-size units called pages. AIX 5.3 ML03 (and later versions of AIX) running on POWER5+ processor supports four page sizes: 4KB, 64KB, 16MB and 16GB. The role of the VMM is to manage the allocation of real-memory page frames and to resolve references by the program to virtual-memory pages that are not currently in real memory or do not yet exist (for example, when a process makes the first reference to a page of its data segment).

Because the amount of virtual memory that is in use at any given instant can be larger than real memory, the VMM must store the surplus on disk. From the performance standpoint, the VMM has two, somewhat opposed, objectives:

- Minimize the overall processor-time and disk-bandwidth cost of the use of virtual memory
- Minimize the response-time cost of page faults

In pursuit of these objectives, the VMM maintains a free list of page frames that are available to satisfy a request for memory. The VMM uses a page-replacement algorithm to determine which virtual-memory pages currently in memory will have their page frames reassigned to the free list. The page-replacement algorithm uses several mechanisms:

- Virtual-memory segments are classified as containing either computational or file memory.
- Virtual-memory pages whose access causes a page fault are tracked.
- Page faults are classified as new-page faults or as repage faults.
- Statistics are maintained on the rate of repage faults in each virtual-memory segment.
- User-tunable thresholds influence the page-replacement algorithm's decisions.

The following sections describe the free list and the page-replacement mechanisms in more detail.

2.1.2. Free List

The VMM maintains a logical list of free page frames that it uses to accommodate page faults. In most environments, the VMM must occasionally add to the free list by reassigning some page frames owned by running processes. The virtual-memory pages whose page frames are to be reassigned are selected by the VMM's page-replacement algorithm. The VMM thresholds determine the number of frames reassigned.

By default in AIX 6.1, and optionally in AIX 5.3 the LRU algorithm can either use lists or the page frame table. Prior to AIX 5.3, the page frame table method was the only method available. The list-based algorithm provides a list of pages to scan for each type of segment.

You can disable the list-based LRU feature and enable the original physical-address-based scanning with the page_steal_method parameter of the vmo command. The default value for the page_steal_method parameter is 1, which means that the list-based LRU feature is enabled and lists are used to scan pages. If the page_steal_method parameter is set to 0, the physical-address-based scanning is used. The value for the page_steal_method parameter takes effect after a `bosboot` and `reboot`.

2.1.3. Persistent versus Working Segments

The pages of a persistent segment have permanent storage locations on disk. Files containing data or executable programs are mapped to persistent segments. Because each page of a persistent segment has a permanent disk storage location, the VMM writes the page back to that location when the page has been changed and can no longer be kept in real memory. If the page has not changed when selected for placement on a free list, no I/O is required. If the page is referenced again later, a new copy is read in from its permanent disk-storage location.

Working segments are transitory, exist only during their use by a process, and have no permanent disk-storage location. Process stack and data regions are mapped to working segments, as are the kernel text segment, the kernel-extension text segments, as well as the shared-library text and data segments. Pages of working segments must also have disk-storage locations to occupy when they cannot be kept in real memory. The disk-paging space is used for this purpose.

The following illustration shows the relationship between some of the types of segments and the locations of their pages on disk. It also shows the actual (arbitrary) locations of the pages when they are in real memory.

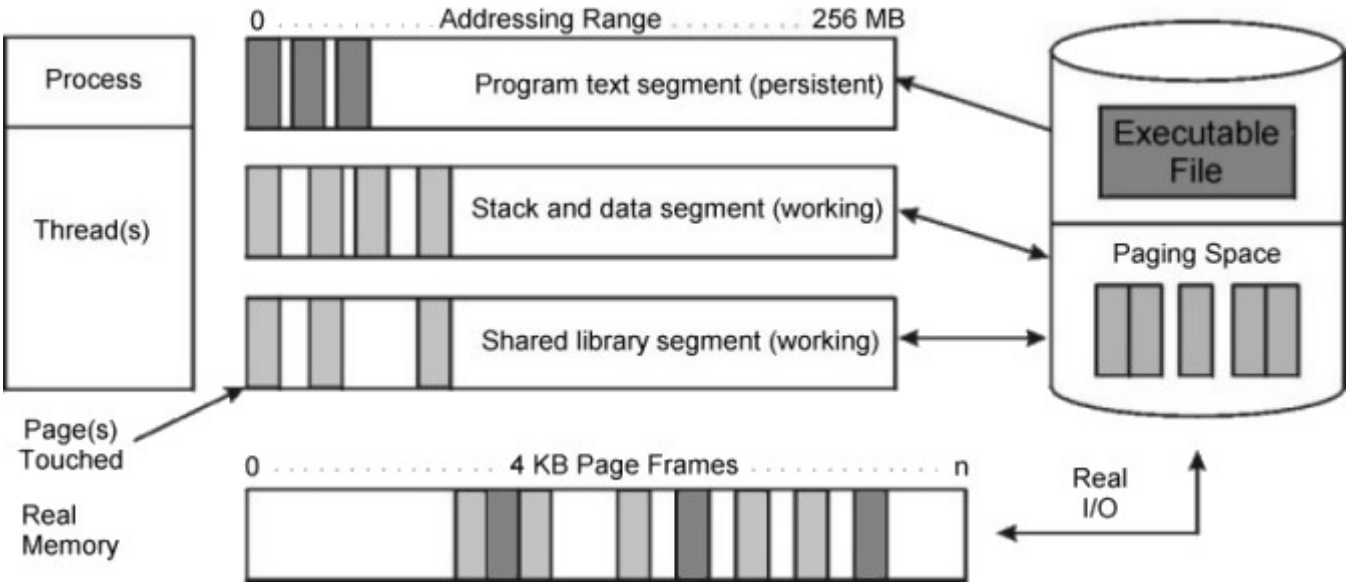


Figure 1-4 Persistent and Working Storage Segments.

Persistent-segment types are further classified.

Client segments are used for:

- mapping remote files (for example, files that are being accessed through NFS), including remote executable programs. Pages from client segments are saved and restored over the network to their permanent file location, not on the local-disk paging space.
- mapping the local files that are located on the JFS2 or Veritas based filesystem

Non-Client segments are used for:

- mapping the local files that are located on the JFS based filesystem

Journaled and deferred segments are persistent segments that must be atomically updated. If a page from a journaled or deferred segment is selected to be removed from real memory (paged out), it must be written to disk paging space unless it is in a state that allows it to be committed (written to its permanent file location).

2.1.4. Computational versus File Memory

Computational memory, also known as computational pages, consists of the pages that belong to working-storage segments or program text (executable files) segments.

File memory (or file pages) consists of the remaining pages. These are usually pages from permanent data files in persistent storage.

2.1.5. Page Replacement

When the number of available real memory frames on the free list becomes low, a page stealer is invoked. A page stealer moves through the Page Frame Table (PFT), looking for pages to steal.

The PFT includes flags to signal which pages have been referenced and which have been modified. If the page stealer encounters a page that has been referenced, it does not steal that page, but instead, resets the reference flag for that page. The next time the clock hand (page stealer) passes that page and the reference bit is still off, that page is stolen. A page that was not referenced in the first pass is immediately stolen.

The modify flag indicates that the data on that page has been changed since it was brought into memory. When a page is to be stolen, if the modify flag is set, a pageout call is made before stealing the page. Pages that are part of working segments are written to paging space; persistent segments are written to disk.

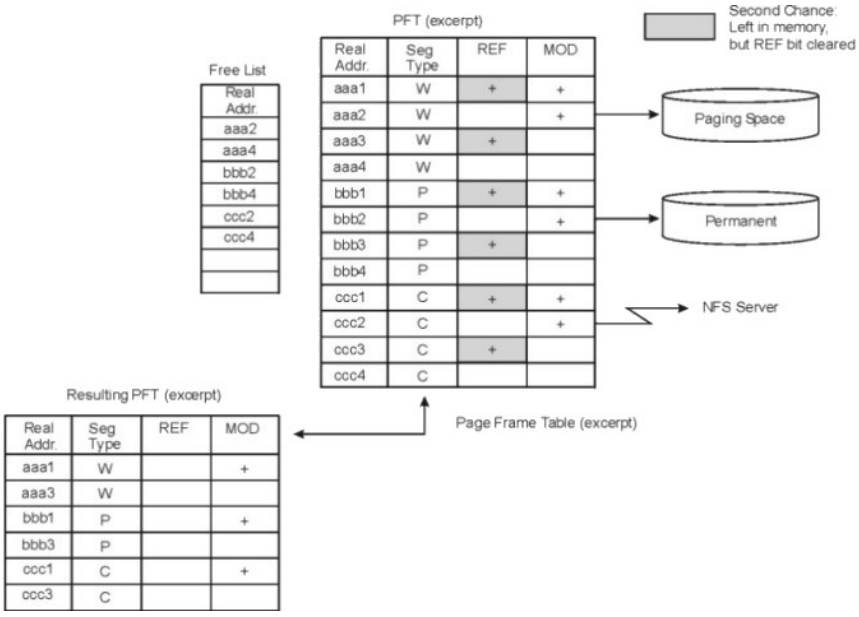


Figure 1-5 Page Replacement Example.

The illustration consists of excerpts from three tables. The first table is the page frame table with four columns that contain the real address, the segment type, a reference flag, and a modify flag. A second table is called the free list table and contains addresses of all free pages. The last table represents the resulting page frame table after all of the free addresses have been removed.

In addition to the page-replacement, the algorithm keeps track of both new page faults (referenced for the first time) and repage faults (referencing pages that have been paged out), by using a history buffer that contains the IDs of the most recent page faults. It then tries to balance file (persistent data) page outs with computational (working storage or program text) page outs.

When a process exits, its working storage is released immediately and its associated memory frames are put back on the free list. However, any files that the process may have opened can stay in memory.

Page replacement is done directly within the scope of the thread if running on a uniprocessor. On a multiprocessor system, page replacement is done through the lrud kernel process, which is dispatched to a

CPU when the minfree threshold has been reached. Starting with AIX 4.3.3, the lrud kernel process is multithreaded with one thread per memory pool. Real memory is split into one or more memory pools based on the number of CPUs and the amount of RAM. The number of memory pools on a system (LPAR) can be determined by running the vmo -L command.

The values for minfree and maxfree in the vmo command output will be the sum of the minfree and maxfree for each memory pool.

2.1.6. Repaging

A page fault is considered to be either a new page fault or a repage fault. A new page fault occurs when there is no record of the page having been referenced recently. A repage fault occurs when a page that is known to have been referenced recently is referenced again, and is not found in memory because the page has been replaced (and perhaps written to disk) since it was last accessed.

A perfect page-replacement policy would eliminate repage faults entirely (assuming adequate real memory) by always stealing frames from pages that are not going to be referenced again. Thus, the number of repage faults is an inverse measure of the effectiveness of the page-replacement algorithm in keeping frequently reused pages in memory, thereby reducing overall I/O demand and potentially improving system performance.

To classify a page fault as new or repage, the VMM maintains a repage history buffer that contains the page IDs of the N most recent page faults, where N is the number of frames that the memory can hold. For example, 512 MB memory requires a 128 KB repage history buffer. At page-in, if the page's ID is found in the repage history buffer, it is counted as a repage. Also, the VMM estimates the computational-memory repaging rate and the file-memory repaging rate separately by maintaining counts of repage faults for each type of memory. The repaging rates are multiplied by 0.9 each time the page-replacement algorithm runs, so that they reflect recent repaging activity more strongly than historical repaging activity.

2.2. Memory and Paging

Several numerical thresholds define the objectives of the VMM. When one of these thresholds is breached, the VMM takes appropriate action to bring the state of memory back within bounds. This section discusses the thresholds that the system administrator can alter through the **vmo** command.

2.2.1. AIX 6.1 Restricted Tunables concept

Since AIX 5.3, six tuning commands (vmo, ioo, schedo, raso, no and nfso) are available to tune the operating system for the specific workload. Beginning with AIX 6.1, some tunables are now classified as restricted use tunables. The default values for restricted tunables are considered to be optimum for most AIX environments and should only be modified at the recommendation of IBM support professionals.

As these parameters are not recommended for user modification, they are no longer displayed by default, but can be displayed with the –F option (force). The –F option forces the display of restricted tunable parameters when the options –a, -L, or –x are specified alone on the command line to list all tunables. When –F is not specified, restricted tunables are not included in a display unless specifically named in association with a display option.

2.2.2. AIX free memory

In AIX, memory is hierarchically represented by the data structures vmpool, mempool, and frameset. A vmpool represents an affinity domain of memory. A vmpool is divided into multiple mempools, each mempool being managed by a single page replacement least recently used (LRU) daemon. Each mempool is further subdivided into one or more framesets that contain the free-frame lists, so as to improve the scalability of free-frame allocators.

The number of page frames on the free list is controlled by the following thresholds and are managed individually for each memory pool:

- **minfree**
  - Minimum acceptable number of real-memory page frames in the free list. When the size of the free list falls below this number, the VMM begins stealing pages. It continues stealing pages until the size of the free list reaches maxfree,
- **maxfree**
  - Maximum size to which the free list will grow by VMM page-stealing. The size of the free list may exceed this number as a result of processes terminating and freeing their working-segment pages or the deletion of files that have pages in memory.

The units used are the numbers of 4k memory pages.

In earlier releases of AIX (5.2 and 5.1), minfree and maxfree values are system wide thresholds, even when there are multiple mempools present. Therefore, the total number of pages on the free list is normally >= minfree.
In AIX 5.3 & 6.1, minfree and maxfree thresholds apply to each mempool individually. Whenever the number of free pages in a given mempool drops below minfree, the LRU daemon for that mempool goes about freeing pages in that mempool until the free page count in that mempool reaches maxfree. Therefore, the number of pages on the free list is normally >= minfree times the number of memory pools.

The minfree and maxfree values are modified using the vmo command. Generally, we want a large enough value for minfree so that there are always a few pages on the free list so we never need to wait for lrud, the process implementing the "page stealing", to execute in order to satisfy a new memory page request. And, the difference between the maxfree and minfree settings should be large enough that lrud is not constantly starting and stopping.

Recommended starting values for minfree and maxfree are being calculated in the following way:

Initial Settings for AIX 5.3 & 6.1	**Initial Settings for AIX 5.2**
minfree = max [960, (lcpus * 120) / # of mempools]	minfree = max (960, lcpus * 120)
$maxfree = minfree + \frac{(Max\ Read\ Ahead * lcpus)}{\#\ of\ mempools}$	maxfree = minfree + (Max Read Ahead * lcpus)
Where,	
Max Read Ahead = max (maxpgahead, j2_maxPageReadAhead)	
Number of memory pools = "echo mempool * | kdb" and count them	

(*) if SMT activated, lcpu = 2 * No. Physical Processors

2.2.3. AIX file system cache size

Excessive paging activity decreases performance substantially. This can become a problem with database files created on journaled file systems (JFS and JFS2). In this situation, a large number of SGA data buffers might also have analogous journaled file system buffers containing the most frequently referenced data. The behavior of the AIX file buffer cache manager can have a significant impact on performance. On AIX, tuning buffer-cache paging activity is possible but it must be done carefully and infrequently.

Optimal file system cache size depends mostly on the workload and I/O characteristics of your database and whether you are using a JFS(2) file system based database or raw devices. If JFS2 DIO or CIO options are active, no file system cache is being used.

The following thresholds are expressed as percentages. They represent the fraction of the total real memory of the machine that is occupied by file pages (pages or non-computational segments):

- **minperm%**
  - If the percentage of real memory occupied by file pages falls below the MINPERM value, the page-replacement algorithm steals both file and computational pages, regardless of repage rates or setting of 'lru_file_repage',
- **maxperm%**
  - If the percentage of real memory occupied by file pages rises above the MAXPERM value, the page-replacement algorithm steals only file pages,
- **maxclient%**
  - If the percentage of real memory occupied by client file pages rises above the MAXCLIENT value, the page-replacement algorithm steals only client pages,
  - maxclient% <= maxperm and maxperm% includes maxclient%,
- **minperm% <> maxperm%**
  - If the percentage of real memory occupied by file pages is between the MINPERM and MAXPERM parameter values, the virtual memory manager (VMM) normally steals only file pages, but if the repaging rate for file pages is higher than the repaging rate for computational pages, then computational pages are stolen as well. When the LRU_FILE_REPAGE is set to zero (0), the repage rate is ignored and only file pages will be stolen.

A simplistic way of remembering this is that AIX will try to keep the AIX buffer cache size between the **minperm%** and **maxperm%** percentage of memory. Use the "**vmo –a (-F)"** command to determine the current **minperm%, maxperm%** and **maxclient%** values. The vmo command shows these values as a percentage of real memory (minperm%, maxperm%, maxclient%) as well as the total number of 4k pages (minperm, maxperm). The "vmstat-v" command may also be used to display minperm, maxperm and maxclient settings as a percentage basis. In addition, "vmstat –v" also shows the current number of file and client pages in memory as well as the percentage of memory these pages currently occupy.
The minperm, maxperm and maxclient threshold percentages can be changed using the vmo parameters minperm%, maxperm% and maxclient%.

With Oracle database workloads, we want to make sure that the computational pages used for Oracle executable code, the Oracle SGA and Oracle PGA, etc. always stay resident in memory. And, since the

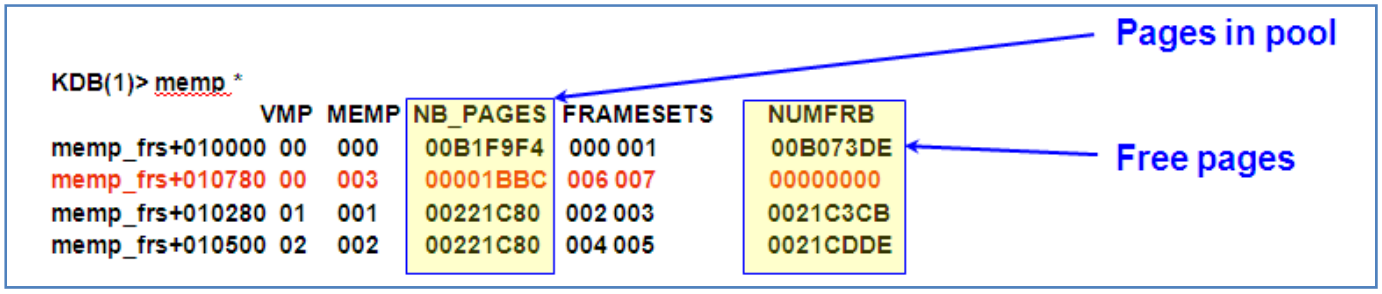
Oracle DB buffer cache already provides caching for Oracle .dbf file data, the AIX file system cache provides a somewhat redundant feature. Therefore, we want to use the Virtual Memory Management (VMM) policies that favor computational pages over file system cache pages.

- **Typical vmo settings for Oracle environments:**
  - **lru_file_repage** = 0 (the default is 1 for AIX 5.2/5.3 and 0 for AIX 6.1)
    - The parameter was introduced at ML4 AIX 5.2 and ML1 AIX 5.3
    - It provides a hint to the VMM about whether re-page counts should be considered when determining what type of memory to steal
  - **lru_poll_interval** = 10 (msecs) (the default is 10)
    - The parameter was introduced at ML4 AIX 5.2 and ML1 AIX 5.3. It tells the page stealer lrud whether it should stop working and poll for interrupts or continue processing until the current request for free frames has been satisfied
  - **minperm%** = 3 (the AIX 5.2/5.3 default is 20)
    - Target for minimum % of physical memory to be used for file system cache
    - minperm% < numperm% (check output of the vmstat –v)
  - **maxperm%** = 90 (the AIX 5.2/5.3 default is 80)
    - Target for maximum % of physical memory to be used for JFS file system cache
    - maxperm% >= maxclient% >= numclient% (check output of the vmstat –v)
    - Prior to AIX 5.3, a large file system cache may cause performance issues. For LPAR's running AIX 5.2 (or earlier) with > 25 GB of physical memory, maxperm% should be reduced so that the target maximum file system cache size is <= 20 GB. For example, if there is a 50 GB of physical memory, maxperm% should be set to 40 or less and strict_maxperm should be set to 1.
  - **strict_maxperm** = 0 (the default)
    - Enables/disables enforcement of maxperm as a "hard" limit
    - Prior to AIX 5.3, a large file system cache may cause performance issues. For LPAR's running AIX 5.2 (or earlier) with > 25 GB of physical memory, maxperm% should be reduced so that the target maximum file system cache size is <= 20 GB. If this is done, strict_maxperm should be set to 1 to enforce a hard limit on maxperm%.
  - **maxclient%** = 90 (the AIX 5.2/5.3 default is 80)
    - Target for maximum % of physical memory to be used for JFS2/NFS file system cache
    - maxclient% should normally be set equal to maxperm%. If maxperm% default value is changed, maxclient% should be changed accordingly.
  - **strict_maxclient** = 1 (the default)
    - Enables/disables enforcement of the maxclient as a "hard" limit (important: value can be changed, but needs to be done very carefully in respect to the memory utilization)
    - With value 0, protects computational memory pages and only file memory pages are stolen

2.2.4. Memory Affinity

IBM POWER-based SMP system contains one or more multichip modules (MCMs), each containing multiple processors. Although any processor can access all of the memory in the system, a processor has faster access when addressing memory that is attached to its own MCM. By enabling the memory affinity, AIX attempts to satisfy a page fault using memory attached to the MCM containing the processor that caused the page fault.
In some cases where multiple LPAR's are defined on the server, the sizes of the memory pools for a given LPAR may be unbalanced. In this situation, the LRU daemon for the smaller memory pool may have to work really hard to try to find the memory pages that can be freed, resulting in high scan/free rates for this specific pool. This can be determined using the vmstat command. In order to make free pages available in this pool, AIX may be forced to page out computational pages, even though there may be lots of free pages available in other memory pools.

To find out if memory pool sizes are unbalanced, use the following command:



In situations where this occurs, the issue can often be resolved by disabling memory_affinity.

On AIX 5.2, 5.3 and 6.1, this can be done using the following command:

vmo –r –o memory_affinity=0

With memory_affinity = 0 AIX sees a flat memory space and memory pools are roughly equal in size. Once the command is executed, the system has to be rebooted. The parameter memory_affinity is a restricted tunable in AIX 6.1, so we recommend working with the AIX Supportline to determine if this is recommended for your environment.

2.2.5. Allocating and using sufficient paging space

Inadequate paging space can result in a process being killed, system hang or AIX crash. On AIX, you dynamically add and resize paging space on raw disk partitions. The amount of paging space you should configure depends on the amount of physical memory present and the paging space requirements of your applications. Good starting point would be a '½ the physical memory + 4 GB' up to the size of a single internal disk. Use the lsps command to monitor paging space use and the vmstat command to monitor system paging activities.

Prior to AIX 4.3.2, paging space needed to be large, typically two to three times the size of real memory. This is because page space was pre-allocated when a program started, whether it used the page space or not. AIX 4.3.2 and higher use deferred paging, where paging space is not allocated until needed. The system uses swap space only if it runs out of real memory. If the memory is sized correctly, there is no paging and the page space can be small. Workloads where the demand for pages does not fluctuate significantly perform well with a small paging space. Workloads likely to have peak periods of increased paging require enough paging space to handle the peak number of pages.

Constant and excessive paging indicates that the real memory is over-committed. In general, you should avoid paging at all.

The following AIX commands provide paging status and statistics:

```
# vmstat –s
# lsps –a
```

2.2.6. Pinning SGA Shared Memory

The primary motivation for considering the pinning of SGA memory is to prevent Oracle SGA from ever being paged out. In a properly tuned Oracle on AIX environment there should not be any paging activity to begin with, so SGA related pages should stay resident in physical memory even without explicitly pinning them. In improperly configured or tuned environments where the demand for computational pages exceeds the physical memory available to them, SGA pinning will not address the underlying issue and will merely cause other computational pages (e.g. Oracle server or user processes) to be paged out. This can potentially have as much or more impact on overall Oracle performance as the paging of infrequently used SGA pages.

Memory access intensive applications that use large amounts of virtual memory may obtain performance improvements by using large pages in conjunction with SGA pinning. The large page related performance improvements are attributable to reduced translation lookaside buffer (TLB) misses. Large pages also improve memory prefetching by eliminating the need to restart prefetch operations on 4KB boundaries.

- **Pinning shared memory**
  - AIX Parameters
    - vmo –p –o v_pinshm = 1 (allow pinning of Shared Memory Segments)
    - leave maxpin% at the default of 80
  - Oracle Parameters
    - `LOCK_SGA = TRUE`
  - Enabling AIX Large Page Support
    - vmo –p –o lgpg_size = 16777216 –o lgpg_regions = number_of_large_pages
    - input calculation: number_of_large_pages = INT [(SGA size – 1) / 16 MB)] + 1
  - Allowing Oracle to use Large Pages
    - chuser capabilities=CAP_BYPASS_RAC_VMM, CAP_PROPAGATE oracle

If not done properly, Oracle SGA pinning and the use of large pages can potentially result is significant performance issues and/or system crashes. And, for many Oracle workloads SGA pinning is unlikely to provide significant additional benefits. It should therefore only be considered where there is known performance issue that could not be addressed through other options, such as VMM parameter tuning.

Following are some general guidelines I suggest for customers considering SGA pinning:

1. There should be a strong change control process in place and there should be good interlock between DBA and AIX admin staff on proposed changes. For example, DBAs should not install new Oracle instances, or modify Oracle SGA or PGA related parameters without coordinating with AIX admin.
2. Prior to SGA pinning, the system pinned page usage (e.g. through svmon or vmstat) should be monitored prior to SGA pinning to verify that pinning is not likely to result in pinned page demand exceeding maxpin%
3. maxpin% should not be changed from its default setting of 80. Any exceptions should undergo critical technical review regarding perceived requirements, workload profile and associated risks.
4. SGA pinning should not be considered if the aggregate SGA requirement (for all instances in the LPAR) exceeds 60% of the total physical memory. This allows for 20% or more of the physical memory to be used for non-SGA related pinning. Any exceptions should undergo critical technical review regarding perceived requirements, workload profile and associated risks.
5. If large pages (16 MB) are used for SGA pinning, any DBA changes to SGA size should be tightly coupled to corresponding AIX admin vmo lgpg_regions changes. An imbalance between SGA size and the lgpg_regions allocation may result in sub-optimal performance.
6. Monitor pinned page usage (e.g. through svmon or vmstat) on an ongoing basis to ensure there is always an adequate reserve of pinnable pages. Consider implementing automated alerts to notify in the event the reserve drops below some specified threshold.
7. Application and system performance should be observed with and without SGA pinning (in a stress test environment if possible). If there are no observable and quantifiable performance improvements due to SGA pinning, SGA pinning should not be used.

2.3. Asynchronous I/O

Oracle takes full advantage of asynchronous I/O (AIO) provided by AIX, resulting in faster database access. Using the LVM and hardware based striping enhances the effectiveness of AIO. The LVM reduces disk contention by striping data across multiple disk spindles. Using AIO with LVM significantly improves RDBMS performance.
AIX versions 4 and higher support asynchronous I/O (AIO) for database files created both on file system partitions and on raw devices. AIO on raw devices is implemented fully in the AIX kernel, and does not require server processes to service the AIO requests. When using AIO on file systems, the kernel server processes (kproc) control each request from the time a request is taken off the queue until it completes. The number of kproc servers determines the number of AIO requests that can be executed in the system concurrently, so it is important to tune the number of kproc processes when using filesystems to store Oracle datafiles. With all AIX versions, if the database is using AIO, the corresponding subsystem must be activated by setting *available* in the autoconfig parameter. This is the default in AIX 6.1. Changing this tunable will require the reboot of the system.
When using AIO on raw devices and disk drives in Oracle ASM environment, IO is being done via the fastpath when the fastpath is enabled (default in 5.3 and 6.1). This hands off the IO to the hdisk driver reducing CPU context switching. Starting with AIX 5.3 fastpath functionality for the CIO mounted filesystems can be enabled by setting the following tunable: fsfastpath = 1. This change is not persistent across the system reboot. In AIX 6.1, now restricted tunable fsfastpath is set to 1 by default.
Set the minimum value to the number of servers to be started at system boot. Set the maximum value to the number of servers that can be started in response to a large number of concurrent requests. These parameters apply to files only; they do not apply to raw devices.

In AIX 5.3, the default value for the minimum number of servers is 1, while the default value for the maximum number of servers is 10. AIX 6.1 scope and the default values are changed: aio_minserver = 3, aio_maxserver = 30 and the scope is per CPU. All parameters in 6.1 have leading aio_ in parameter name.

If the value of the maxservers parameter is set too low, you might see the following error messages repeated:

Warning: lio_listio returned EAGAIN
Performance degradation may be seen.

In addition to changing the tunables minservers and maxservers for AIO, the maxreqs tunable should also be monitored and changed, if necessary. The maxreqs tunable controls the number of requests the AIO system allows and it is applicable for both (fs)fastpath and kproc based I/O operations.

To display the number of AIO servers running, enter the following command as the root user:

```
# pstat -a | grep -c aios | wc –l
```

AIX 5.3 brings new option for the iostat command (iostat –A) which displays AIO statistics for the specified interval and count. With AIX 6.1, all AIO subsystems parameters became tunable via the ioo command (aioo command is deprecated).

The asynchronous I/O report has the following column headers:

- avgc: Average global AIO request count per second for the specified interval,
- avfc: Average fastpath request count per second for the specified interval,
- maxgc: Maximum global AIO request count since the last time this value was fetched,
- maxfc: Maximum fastpath request count since the last time this value was fetched,
- maxreqs: Maximum AIO requests allowed.

Recommended starting values:

- **minservers:**
  - entry value for AIX 5.3 is defined per system, starting value should be at least 100
  - entry value for AIX 6.1 is defined per CPU, starting value should be at least 3 (aio_minservers)
- **maxservers:**
  - entry value for AIX 5.3 is defined per CPU, starting value should be at least 100
  - entry value for AIX 6.1 is defined per CPU, starting value should be at least 30 (aio_maxservers)
- **maxreqs:**
  - a multiple of 4096 > 4 * number of disk * disk_queue_depth, typical setting is 16384, or 65536 for AIX 6.1 (aio_maxreqs)

Higher numbers will not hurt performance, as it only results in more kernel processes running that do not actually get used.

Use one of the following commands to set the number of servers and queue depth:

  - in AIX 5.X,
    - "smit aio" AIX management panel,
    - chdev -P -l aio0 -a maxservers='*m*' -a minservers='*n*',
    - chdev -P -l aio0 -a maxreqs='q',
  - in AIX 6.1,
    - ioo command is used to change all AIO related tunables.
    - chdev -P -l aio0 -a aio_maxservers='*m*' -a aio_minservers='*n*',
    - chdev -P -l aio0 -a aio_maxreqs='q',

Additionally, the following set of the Oracle parameters should also be checked. Presented are the default values:

- `DISK_ASYNC_IO=TRUE,`
- `FILESYSTEMIO_OPTIONS=(ASYNCH | SETALL),`
- `DB_WRITER_PROCESS` = usually left at the default value.

2.4. I/O Configuration

The I/O subsystem is one of the most important components of the Oracle RDBMS based system. In many occasions, the performance of applications will be limited by the disk I/O. If the application is spending most of the CPU time waiting for I/O to complete, it is said to be I/O bound. This means that the I/O subsystem is not able to service the I/O requests within an acceptable time.
To avoid this situation, a proper storage subsystem and data layout planning is a mandatory prerequisite before deploying the Oracle database. The most general rule to achieve optimal performance is evenly distributing I/O load across all available elements of the storage subsystem(s).

This chapter is only focusing on the subset of the AIX IO performance tunables that are directly affecting the activity of the Oracle database.

2.4.1. AIX LVM - Volume Groups

A Volume Group is a logical storage structure consisting of the physical volumes that users see as a single storage volume. AIX LVM is used for allocation management of the basic storage elements (volume groups, physical volumes, logical volumes, file systems etc).
There are 3 different Volume Group types supported starting with AIX 5.3: normal, big and scalable VG's. Following figure shows the configuration limits for different VG's:

VG type	Maximum PVs	Maximum LVs	Maximum PPs per VG	Maximum PP size
Normal VG	32	256	32,512 (1016 * 32)	1 GB
Big VG	128	512	130,048 (1016 * 128)	1 GB
Scalable VG	1024	4096	2,097,152	128 GB

We recommend the Scalable VG's for allocation of the Oracle database.
Appendix A - "Configuring IBM System Storage DS4000 Series For Oracle Database Applications" will provide more info about this recommendation (Appendix A).

2.4.2. AIX LVM - Logical Volumes

The AIX Logical Volume Manager (LVM) can stripe data across multiple disks to reduce disk contention. Effective use of the striping features in the LVM allows you to spread I/O more evenly across disks, resulting in greater overall I/O performance.

AIX LVM supports two striping techniques:

- PP (Physical Partition) Spreading
- LVM Striping

PP Spreading is initiated during the creation of the logical volume itself. The result is a logical volume that is spread across multiple hdisks in the Volume Group. To create a LV with PPs spread equally over a set of hdisks in a VG use the following AIX command:

mklv –e x …

If for some reason, actual LVM striping has to be implemented, my advice is to use large grained LVM's.

Strip size should be at least 1MB and not less than the value determined by the following formula:

`DB_BLOCK_SIZE * DB_FILE_MULTIBLOCK_READ_COUNT (usually >= 1MB in size).`

- in many cases db_block_size * db_file_multiblock_read_count will be considerably less than 1MB, so the strip size should be at least as large as is dictated by this formula. For example, for 11i E-Business Suite environments, the recommended settings are db_block_size=8k, db_file_multiblock_read_count=8 == 64k max i/o transfer size.

There may be rare exceptions where a smaller strip size is warranted, e.g. where EXCEPTIONALLY HIGH single threaded sequential read/write performance is needed.

Valid LV stripe sizes are:

- AIX 5.2: 4k, 8k, 16k, 32k, 64k, 128k, 256k, 512k, 1 M
- AIX 5.3/AIX 6.1: AIX 5.2 Strip sizes + 2M, 4M, 16 M, 32M, 64M, 128M

Important assumption/prerequisite for this process is that the underlying storage LUN's have been created using typical RAID technologies (RAID-10 or RAID-5).
Using both of these techniques (SW & HW striping) will provide an optimal storage layout for any Oracle database.

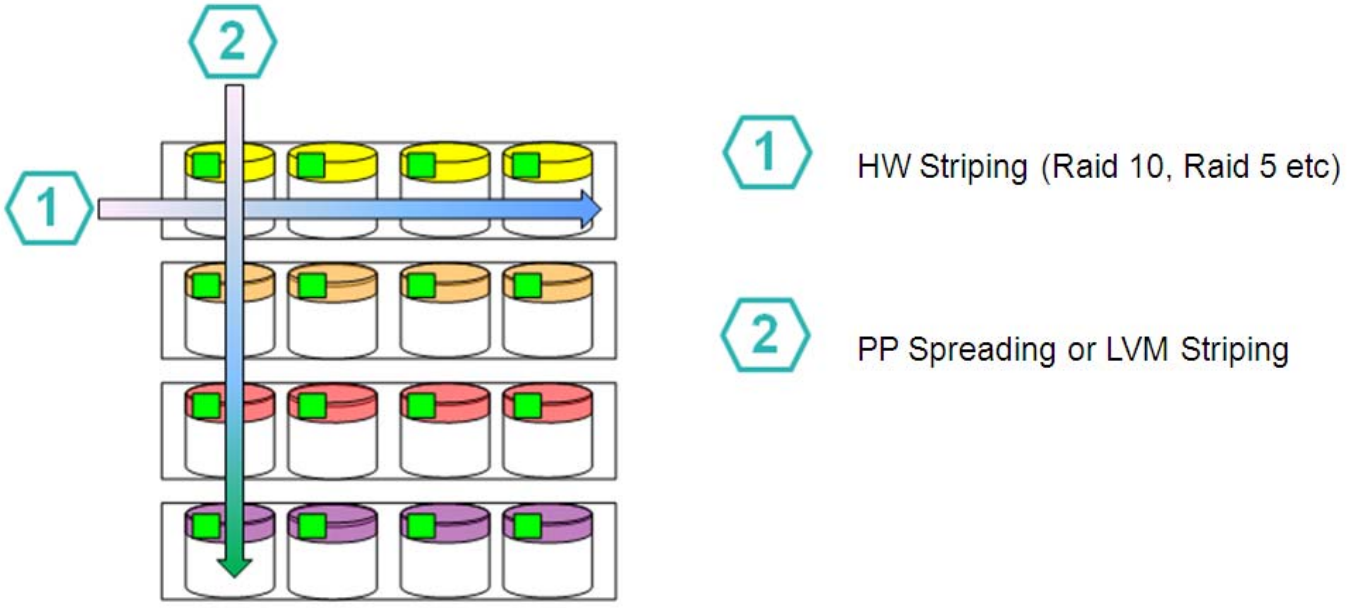


Figure 1-7 Optimal Storage Layout

There is at least one clear advantage of PP spreading over LVM striping. If the LV created using the PP spreading technique has to be extended, any number of disk drives (hdisks) can be added. Once the VG is extended, optimal PP re-distribution of the new disk layout is achieved using the 'reorgvg' command.
Extending the LV that was created using the LVM striping is a more complex process. The number of drives that have to be added to the VG has to be equal to the stripe width that the LV was originally defined and created with.

If raw devices are being used, the AIX Logical Volumes should be created with the Zero Offset Feature. By default, when regular or Big Volume Groups are used, the first 4k bytes of each AIX Logical Volume are reserved for the Logical Volume Control Block (LVCB), with Oracle data beginning at the 2nd 4k block in the Logical Volume. Beginning with Oracle9i Release 2, it is possible to go to a zero offset format where Oracle data begins at byte zero (0) of the Logical Volume. When a db_block_size > 4k is used, using the zero offset option can improve I/O subsystem performance, particularly where some form of striping (e.g. AIX LVM or hardware based RAID-5 or RAID-10) is used.
Currently, the capability to do this is delivered in two parts: An IBM AIX e-fix (APAR IY36656 for AIX 5.1) and an Oracle patch (bug 2620053). Depending on the AIX and Oracle9i levels installed, this functionality may already be included.
Once the prerequisite software has been installed, do the following to take advantage of the zero offset feature:

- create a "big" or "scalable" Volume Group using the mkvg –B or –S flag,
- If a "big" Volume Group is being used, create one or more Logical Volumes in that Volume group using the mklv –T O (not 0) flag. The "-T O" option indicates to Oracle that it can safely use a 0 offset for this Logical Volume. . Logical Volumes created in Scalable Volume Groups will automatically have the "-T O" attribute.

2.4.3. AIX sequential read ahead

The Virtual Memory Manager (VMM) anticipates the need for pages of a sequential file. It observes the pattern in which a process accesses a file. When the process accesses two successive pages of the file, the VMM assumes that the program will continue to access the file sequentially, and schedules additional sequential reads of the file. These reads overlap the program processing and make data available to Oracle sooner.

Two VMM thresholds, implemented as kernel parameters, determine the number of pages it reads ahead:

- **MINPGAHEAD** (JFS) or **j2_minPageReadAhead** (JFS2), the number of pages read ahead when the VMM first detects the sequential access pattern,
  - default: 2,
  - recommended starting value is: MAX (2, `DB_BLOCK_SIZE/4096`),
  - example: ioo –p –o j2_minPageReadAhead=8
  - restricted tunable in AIX 6.1
- **MAXPGAHEAD** (JFS) or **j2_maxPageReadAhead** (JFS2) is the maximum number of pages that VMM reads ahead in a sequential file,
  - default: 8 (JFS) or 128 (JFS2),

- recommended value is the equal to (or multiple of) size of the largest Oracle I/O request,
- `max[256, (DB_BLOCK_SIZE/4096) * DB_FILE_MULTIBLOCK_READ_COUNT]`
- example: ioo –p –o j2_maxPageReadAhead=256
- restricted tunable in AIX 6.1

Set the MINPGAHEAD and MAXPGAHEAD (and equivalent jfs2) parameters to appropriate values for your application (a power of two). Use the **ioo** command to change these values. You can use higher values for the MAXPGAHEAD parameter in systems where the sequential performance of striped logical volumes is of paramount importance.

2.4.4. File system buffer tuning

The LVM uses a construct named pbuf to control a pending disk I/O. A Pbuf is a pinned memory buffer. The LVM always uses one pbuf for each individual I/O request, regardless of the amount of data that is transferred. In previous AIX releases, the pbuf pool was a system wide resource, but the LVM assigns and manages one pbuf pool per VG since AIX 5.3.
To clarify the pbuf related tuning steps we will use the sample result of the following command:

vmstat –v | tail -5 (we only need last 5 lines)

0 pending disk I/Os blocked with no pbuf
- for pbufs, increase pv_min_pbuf using ioo, requires volume group vary-off and on

0 paging space I/Os blocked with no psbuf
- for psbufs, stop paging or add more paging spaces, and preferably tune to stop paging,

8755 filesystem I/Os blocked with no fsbuf ← JFS
- for fsbufs, increase numfsbufs using ioo, default is 196, recommended value is 1568,
- restricted tunable in AIX 6.1

0 client filesystem I/Os blocked with no fsbuf ← NFS/Veritas
- for client filesystem fsbufs, increase:
  - nfso's nfs_vX_pdts and nfs_vX_vm_bufs (where X depends of the NFS version being used, 2, 3 or 4)
- restricted tunables in AIX 6.1

2365 external pager filesystem I/Os blocked with no fsbuf ← JFS2
- for external pager fsbufs, increase:
  - j2_nBufferPerPagerDevice, default is 512, recommended value is 2048,
  - j2_dynamicBufferPreallocation using ioo.
- restricted tunable in AIX 6.1

Every time a change is applied, the affected file system has to be re-mounted.

Final remarks:

- It is important for the reader to understand that the exact counts don't matter here. What is important is how quickly values change over time, so you really need to look at the consecutive vmstat –v results over some period of time – not a single vmstat –v result.
- in AIX 5.2 (or earlier) tuning the pbufs affected system wide resources (pv_min_pbuf), and required volume group vary-off and on again
- in AIX 5.3 these activities became more granular and we can now define them on the VG level using the lvmo command.

2.4.5. JFS2 filesystem DIO/CIO mount options

- **Direct IO (DIO)**

Certain classes of applications (and related DB activities) derive no benefit from the file buffer cache. Databases normally manage data caching at the application level, so they do not need the file system to implement this service for them. The use of a file buffer cache results in undesirable overhead in such cases, since data is first moved from the disk to the file buffer cache and from there to the application buffer. This "double-copying" of data results in additional CPU consumption. Also, the duplication of application data within the file buffer cache increases the amount of memory used for the same data, making less memory available for the application, resulting in additional system overhead due to memory management.

For applications that wish to bypass the buffering of memory within the file system cache, Direct I/O is provided as an option in JFS. When Direct I/O is used for a file, data is transferred directly from the disk to the application buffer, without the use of the file buffer cache. While direct I/O may benefit some applications, its use disables any file system mechanism that depends on the use of file system cache, e.g. sequential read-ahead. Applications that perform a significant amount of sequential read I/O may experience degraded performance when Direct I/O is used.

JFS2 supports Direct I/O as well as concurrent I/O (discussed below) options. The Concurrent I/O model is built on top of the Direct I/O model. For JFS2 based environments, Concurrent I/O should always be used (instead of Direct I/O) for those situations where the bypass of filesystem cache is appropriate.

JFS Direct I/O should only be used against Oracle data (.dbf) files for the environments where the DB_BLOCK_SIZE is 4k or greater. Use of JFS Direct I/O on any other files, including redo logs or control files is likely to result in a severe performance penalty due to violation of Direct I/O alignment and/or I/O transfer size restrictions.

- **Concurrent IO (CIO)**

The inode lock imposes write serialization at the file level. JFS2 (like JFS) is a POSIX compliant filesystem. As such, JFS2 (by default) employs I/O serialization mechanism to ensure the integrity of data being updated. An inode lock is used to ensure that there is at most one outstanding write I/O to a file, reads are not allowed because they may result in reading stale data. Oracle database implements its own I/O serialization mechanisms to ensure data integrity. Consequently, they do not rely on the filesystem

based (POSIX standard) I/O serialization mechanisms. In write intensive environments inode serialization hinders performance by unnecessarily serializing non-competing data accesses.

For applications such as Oracle Database that provide their own I/O serialization mechanisms, JFS2 (beginning with AIX 5.2.10) offers the Concurrent I/O (CIO) option. Under Concurrent I/O, multiple threads may simultaneously perform reads and writes on a shared file. Applications that do not enforce serialization for accesses to shared files (including operating system level utilities) should not use Concurrent I/O, as this could result in data corruption due to competing accesses and /or severe performance penalties.

Concurrent I/O should only be used for Oracle .dbf files (data & index, rbs or undo, system and temp) online redo logs and/or control files. When used for online redo logs or control files, these files should be isolated in their own JFS2 filesystem(s) that have been created with agblksize=512. Filesystem containing .dbf files should be created with agblksize=2048 if DB_BLOCK_SIZE=2k, or agblksize=4096 if DB_BLOCK_SIZE>= 4k. Failure to implement these agblksize guidelines is likely to result in a severe performance penalty. Do not under any circumstances, use CIO mount option for the filesystem containing the Oracle binaries (!!!). Additionally, do not use DIO/CIO options for filesystem containing archive logs or any other files not already discussed.

For applications that wish to bypass the buffering of memory within the filesystem cache, Concurrent I/O is the preferred option for JFS2. When concurrent I/O is used for a file, data is transferred directly from the disk to the application buffer, without the use of the file buffer cache. In addition, the POSIX based I/O serialization mechanisms are disabled, providing better I/O concurrency for write intensive applications. While concurrent I/O may benefit some applications, its use disables any file system mechanisms that depend of the use of file system cache, e.g. sequential read-ahead. Applications that perform a significant amount of the sequential read I/O may experience degraded performance when Concurrent I/O is used.

Applications that use raw logical volumes for data storage don’t encounter inode lock contention since they don’t access files.

2.5. CPU Tuning

CPU tuning we define as an advanced tuning area. Most of the activities are primarily controlled using AIX **schedo** command. Our recommendation here would be to leave all the initial settings of the **schedo** parameters at their default value. Change to any of these parameters should be applied only if specific recommendation has been given by IBM or Oracle.

For that reason, I will focus primarily on the new features of the Power Systems POWER5 & 6 technologies: 'Simultaneous Multi-Threading (SMT)', 'Logical Partitioning' and 'Micropartitioning'.

2.5.1. Simultaneous Multi-Threading (SMT)

Simultaneous multi-threading (SMT) is a hardware design enhancement in POWER5 and later that allows two separate instruction streams (threads) to execute simultaneously on the processor. AIX 5L Version 5.3 or later is also required.

For most multi-threaded commercial workloads, SMT (vs. no-SMT) provides a significant increase in the total workload throughput that is achievable given the same number of physical processors in a partition. The use of SMT is generally recommended for all Oracle workloads. Some applications such as cache or memory intensive High Performance Computing workloads that are tuned to optimize the use of processor resources may actually see a decrease in performance due to increased contention to cache and memory. Single threaded applications (running at relatively low processor utilization) may also see some response time degradation. SMT may be disabled for these cases.

Each hardware thread is supported as a separate logical processor by AIX 5L Version 5.3, independent of the type of partition. When SMT is enabled, a partition with one dedicated processor would have 2 logical processors, a shared partition with 2 virtual processors would have 4 logical processors, etc. Both active threads running on an SMT processor are always from the same dedicated or virtual processor.

The SMT policy is controlled by the operating system, thus it is partition specific and can be enabled or disabled for a given partition. This is done using the **smtctl** command, which controls the enabling and disabling of SMT mode. It provides privileged users and applications a means to enable or disable SMT for all processors in a partition either immediately or on a subsequent boot of the system.

Attached are 3 "NMON" generated diagrams that present performance measurement of the Power System & Oracle based customer system that tested the impact of SMT on the overall performance of the system. They are based on the OLTP workload that was executed at the customer location.

- Test #1 description:

On AIX 5.2 system (SMT not supported), the workload was ramped up to 700 virtual users, peaking at 180 transactions per second, with good response time (less than .05 second; typically around .03 second). The environment was CPU constrained as is visible from the attached NMON diagram.

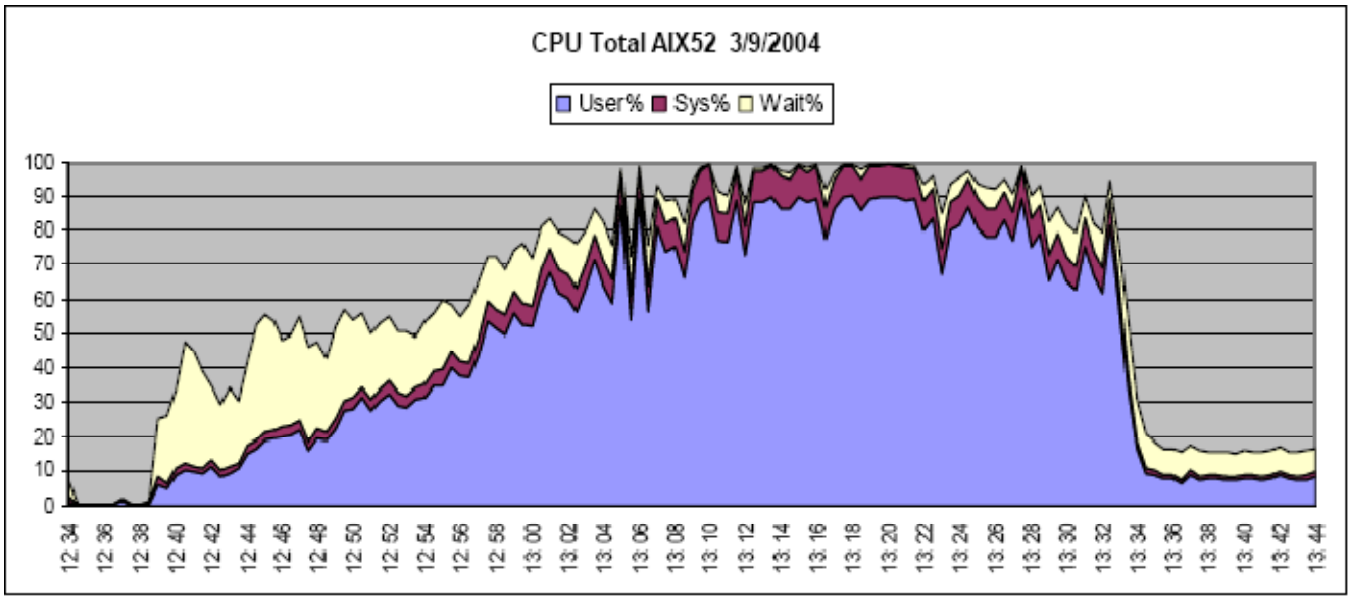


Figure 1-8 CPU utilization of the system with the AIX 52 (no SMT)

- Test #2 description:

Using the same identical hardware configuration, the workload was once again ramped up to 700 virtual users, this time using AIX 53 with SMT enabled. The CPU usage on the AIX 5.3 system now peaked at 70% for the same workload, achieving the same number (180) of transactions per second, with no appreciable increase in response time.

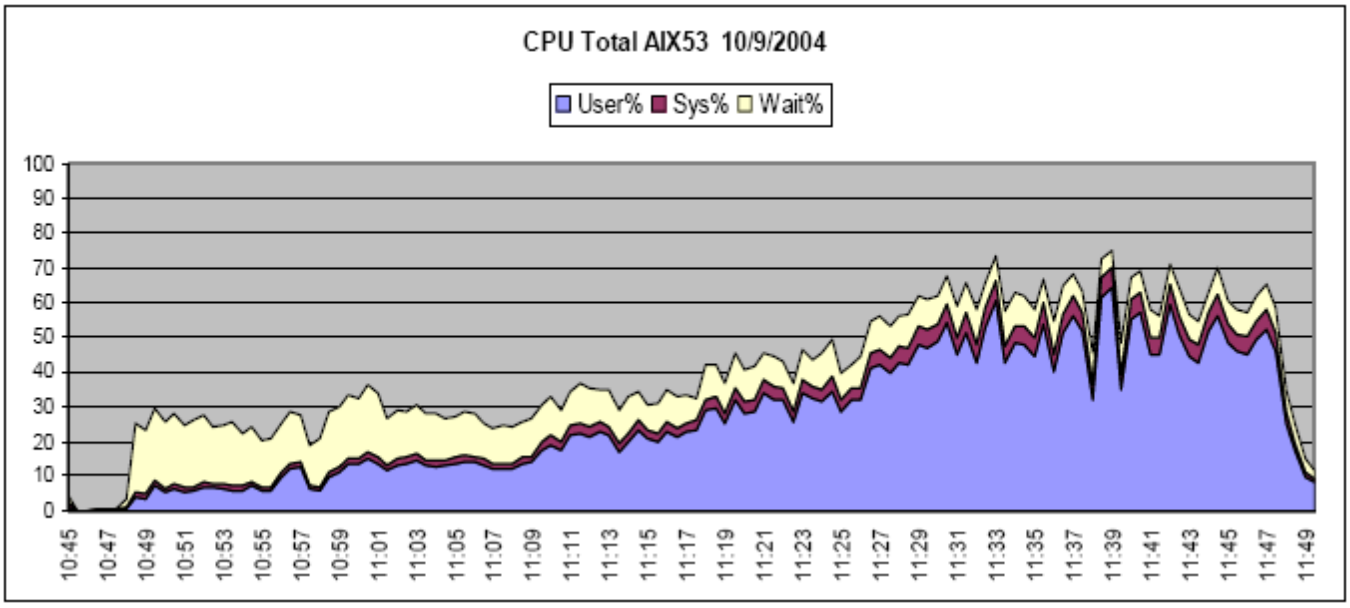


Figure 1-9 CPU utilization of the system with the AIX 5.3 (SMT turned on) and user count of 700

- Test #3 description:

As there was clearly additional capacity available on this server, the customer then increased the maximum number of virtual users to 1000. The increased user count was able to drive a throughput of 280 transactions per second, still with acceptable response time. As shown below, the system had still not reached capacity:

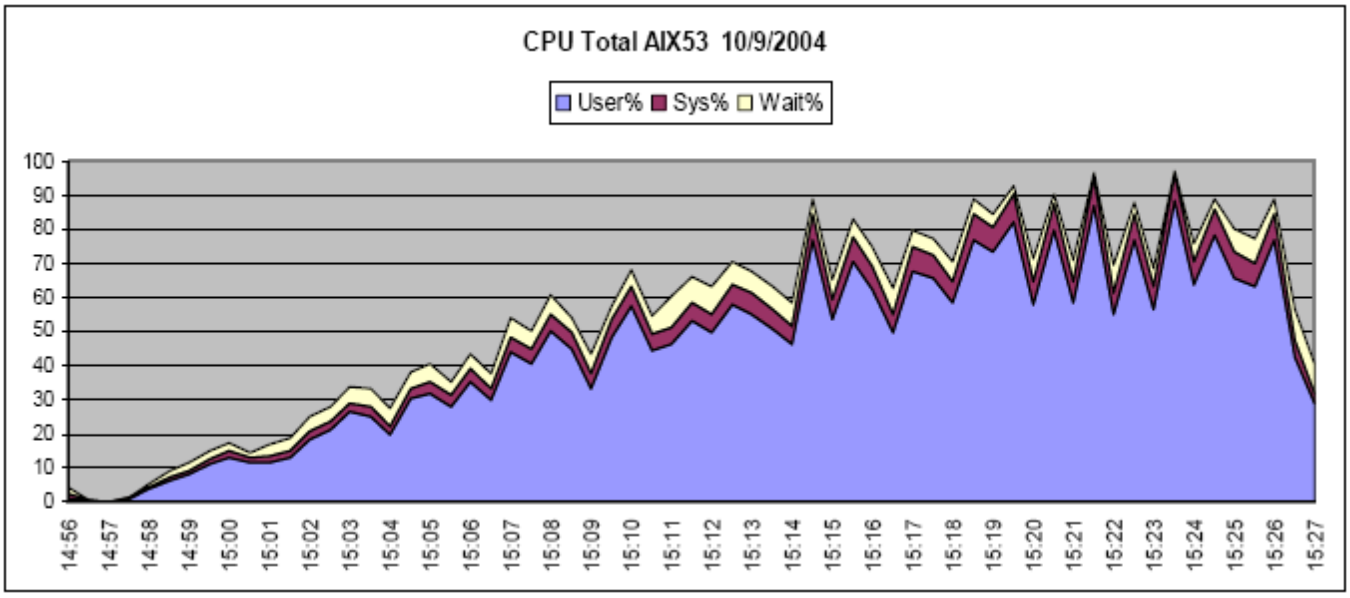


Figure 1-10 CPU utilization of the system with the AIX 5.3 (SMT turned on) and user count of 1000

- Conclusion:

This particular Oracle OLTP test went from 180 transactions per second without SMT to 280 transactions per second with SMT, and from 700 users to over 1000 users, without a significant change in response times.
Results from using SMT vary greatly depending on the type of workload, but we can conclude that Oracle OLTP workloads which are CPU bound will see a significant increase in overall capacity by using the SMT feature of AIX 5.3.

2.5.2. Logical Partitioning

Logical partition in an IBM Power Systems is a subset of the hardware of the SMP that can host an operating system instance. The first release of LPAR support was static in nature, that is, the reassignment of a resource from one LPAR to another LPAR cannot be made while AIX is actively running. As of 2002, IBM Power Systems supports the dynamic reassignment of resources across LPARs running AIX. DLPAR in a IBM Power Systems system, offering a great deal of flexibility to users, and allowing resources to be shifted to where they are most needed without impacting system availability.
Oracle 9i introduced dynamic memory management that when combined with the DLPAR features, partially simplified management of the Oracle instances. Oracle memory utilization primarily consists of memory allocated by the SGA and PGA. The initial size of the SGA memory region is defined by utilizing the variable SGA_MAX_SIZE. Once the database is started (and SGA initialized), certain memory areas can be re-adjusted using the standard Oracle system commands. The management of the PGA area is completely automatic as long as the initial size is defined by utilizing the variable PGA_AGGREGATE_TARGET. The size of this memory area can also be changed dynamically. More details can be found in Chapter 5.
AIX based DLPAR functionality, combined with the new dynamic memory management feature of Oracle 9i, 10g and 11i, provides greater flexibility in the system management of the memory allocation.
DLPAR can also dynamically change the number of CPUs in the system. The Oracle variable CPU_COUNT reflects the number of the CPUs which are enabled in the LPAR. The value of the CPU_COUNT variable directly impacts several different Oracle parameters. The most important are: parallel_max_servers, fast_start_parallel_rollback, db_block_lru_latches, log_buffer etc… Unfortunately, Oracle 9i is not capable of dynamically updating the value of the CPU_COUNT if the number of CPU(s) in the LPAR was changed during the dynamic reconfiguration. However, the AIX scheduler can take advantage of the additional processors. In the case of processor removal, as long as the processor being removed is not bound by any process, it can be removed by the DLPAR operation.
Oracle 10g and 11i will automatically update the value of the CPU_COUNT variable if the value of the CPU(s) in the LPAR has been changed since the database was originally initialized.

2.5.3. Micro-Partitioning

Micro-Partitioning is a feature provided by PowerVM (previously known as APV). It provides the ability of sharing physical processors between logical partitions. The amount of processor capacity that is allocated to a micro-partition (its entitled capacity) may range from ten percent (10%) of a physical processor up to the entire capacity of the shared processor pool. The power hypervisor firmware manages

this feature. It schedules the virtual CPUs on the physical processors. The shared processor pool can have from one physical processor up to the total of the installed processor capacity of the system. Changes to the entitled capacity of a micro-partition can be as granular as one percent (1%) of a physical processor.
Micro-partitions can be defined as:

- Capped

A capped micro-partition has a defined processor entitled capacity which it is guaranteed to receive. There are no circumstances under which a capped micro-partition will receive any more processor resource than its entitled capacity.

- Uncapped

An uncapped micro-partition has a defined processor entitled capacity which it is guaranteed to receive. However, under some circumstances it can receive additional processor resources. Typically, if the uncapped micro-partition is runnable (has real work to execute) and there are unused processor cycles within the physical shared processor pool, then additional cycles can be allocated to the micro-partition on a weighted basis. Capped and uncapped micro-partitions can coexist and both receive the processor resources from the physical shared-processor pool.
Following are some general guidelines we suggest for customers considering running Oracle DB in the Micro-Partitioning environment:

- SMT should be enabled.
- Virtual CPUs for a partition should not exceed the number of physical processors in the shared pool.
- Entitled capacity should be set to the minimum required to meet service level agreements, based on the application workload sizing.
- For capped partitions the number of Virtual CPUs should be set to the nearest integer >= the capping limit. For example, if the capping limit is 3.4, the number of VCPUS should be 4.
- For uncapped partitions, the number of Virtual CPUs should be set to the maximum anticipated peak demand requirement for that partition. It may be appropriate to establish guidelines on the maximum number of Virtual CPUs per CPU of entitled capacity (e.g. no more than 2 or 3 times). This guideline should be periodically reviewed and adjusted based on actual experience as appropriate.
- Where possible, in order to get the highest possible hardware utilization, partitions defined within a single shared pool (server) should exhibit a mix of workload characteristics such as:
  - Capacity requirements peak at different times (e.g. online vs. batch)
  - Service level requirements vary (e.g. high priority production vs. low priority development/test or response sensitive online vs. response insensitive batch)
- The shared processor pool should be sized so that the peak shared pool utilization does not exceed 90%. Shared pool utilization should be monitored on an ongoing basis.
- For batch only applications, a minimum of 2 concurrent batch threads should be run per Virtual CPU (with SMT, there are 2 logical processors per virtual processor). Depending on the application ratio of CPU to I/O wait time, more than 2 concurrent threads per Virtual CPU may be required to utilize all the available CPU resources.

3. Network Tuning

- **UDP Tuning**

Oracle9i and 10g Real Application Clusters on AIX uses User Datagram Protocol (UDP) for interprocess communications. UDP kernel settings should be tuned to improve Oracle performance.
To do this, use the network options (**no**) command to change udp_sendspace and udp_recvspace parameters.

- Set the value of the **udp_sendspace** parameter to [(`DB_BLOCK_SIZE * DB_FILE_MULTIBLOCK_READ_COUNT) + 4096`], but not less than 65536.
- Set the value of the **udp_recvspace** parameter to be >= 4 * udp_sendpace
- If necessary, increase sb_max parameter (default value is 1048576), since sb_max must be >= udp_recvspace

The value of the udp_recvspace parameter should be at least four to ten times the value of the udp_sendspace parameter because UDP might not be able to send a packet to an application before another packet arrives.
To determine the suitability of the udp_recvspace parameter settings, enter the following command:

- netstat -s | grep “socket buffer overflows”

If the number of overflows is not zero, increase the value of the udp_recvspace parameter.

- **TCP Tuning**

TCP set of the parameters that should be changed using the **no** command are:

- rfc1323 = 1
- tcp_sendspace >= 262144
- tcp_recvspace >= 262144
- sb_max >= 1MB (1048576)

4. Oracle Tuning

In this chapter, I will cover the most important items that will make the Oracle database run well on the AIX platform. Besides the presentation of the relatively small sub-group of the Oracle internal parameters I will not be covering details of the internal Oracle tuning activities (index & optimizer related activities, SQL explaining, tracing and tuning etc).
If your are running an ISV packaged application, such as Oracle E-Business Suite or SAP R/3, the package vendor may provide guidelines on basic Oracle setup and tuning. If so, please refer to the appropriate vendor documentation first.

- **Basic Oracle parameters**

There are a limited number of all the Oracle parameters that have a large impact on performance. Without these being set correctly, Oracle cannot operate properly or give good performance. These need to be checked before further fine tuning. It is worth tuning the database further only if all these top parameters are properly defined.
The parameters that make the biggest difference (and should, therefore, be investigated in this order) are listed in the following sections.

- **`DB_BLOCK_SIZE`**

Specifies the default size of Oracle database blocks. This parameter cannot be changed after the database has been created, so it is vital that the correct value is chosen at the beginning. Optimal `DB_BLOCK_SIZE` values vary depending on the application. Typical values are 8 KB for OLTP workloads and 16KB to 32KB for DSS workloads. If you plan to use a 2KB `DB_BLOCK_SIZE` with JFS2 filesystems, be sure to create the filesystem with agblksize=2048.

While most customers only use the default database block size, it is possible to use up to 5 different database block sizes for different objects within the same database. Having multiple database block sizes adds administrative complexity and (if poorly designed and implemented) can have adverse performance consequences. Therefore, using multiple block sizes should only be done after careful planning and performance evaluation.

Some possible block size considerations are as follows:

- Tables with a relatively small row size that are predominantly accessed 1 row at a time may benefit from a smaller `DB_BLOCK_SIZE`, which requires a smaller I/O transfer size to move a block between disk and memory, takes up less memory per block and can potentially reduce block contention.
- Similarly, indexes (with small index entries) that are predominantly accessed via a matching key may benefit from a smaller `DB_BLOCK_SIZE`.
- Tables with a large row size may benefit from a large `DB_BLOCK_SIZE`. A larger `DB_BLOCK_SIZE` may allow the entire row to fit within a block and/or reduce the amount of wasted space within the block.

- Tables or indexes that are accessed sequentially may benefit from a larger DB_BLOCK_SIZE, because a larger block size results in a larger I/O transfer size and allows data to be read more efficiently.
- Tables or indexes with a high locality of reference (the probability that once a particular row/entry has been accessed, a nearby row/entry will subsequently be accessed) may benefit from a larger DB_BLOCK_SIZE, since the larger the size of the block, the more likely the nearby row/entry will be on the same block that was already read into database cache."

- **DB_BLOCK_BUFFERS or DB_CACHE_SIZE**

These parameters determine the size of the DB buffer cache used to cache database blocks of the default size (DB_BLOCK_SIZE) in the SGA. If DB_BLOCK_BUFFERS is specified, the total size of the default buffer cache is DB_BLOCK_BUFFERS x DB_BLOCK_SIZE. If DB_CACHE_SIZE is specified, it indicates the total size of the default buffer cache.

If multiple different block sizes are used within a single database, the DB_nK_CACHE_SIZE parameter is used to specify the size of additional DB cache areas used for blocks that are different from the default DB_BLOCK_SIZE. For example, if DB_BLOCK_SIZE=8K and DB_16K_CACHE_SIZE=32M, this indicates that a separate 32 Megabyte DB cache area will be created for use with 16K database blocks.

The primary purpose of the DB buffer cache area(s) is to cache frequently used data (or index) blocks in memory in order to avoid or reduce physical I/Os to disk when satisfying client data requests. In general, you want just enough DB buffer cache allocated to achieve the optimum buffer cache hit rate. Increasing the size of the buffer cache beyond this point may actually degrade performance due to increased overhead of managing the larger cache memory area. The "Buffer Pool Advisory" section of the Oracle Statspack or Workload Repository (AWR) report provides statistics that may be useful in determining the optimal DB buffer cache size.

- **DISK_ASYNC_IO**

AIX fully supports asynchronous I/O for filesystems (JFS, JFS2, GPFS, Veritas) as well as raw devices. Many sites do not know this fact and fail to use this feature. This parameter should always be set to TRUE (the default value). In addition, the AIX asynchronous I/O feature must be enabled and properly tuned (in AIX 6.1 enabled by default).

- **FILESYSTEMIO_OPTIONS**

This parameter can enable or disable asynchronous I/O or direct I/O on file system files. FILESYSTEMIO_OPTIONS can be set to one of the following values:

- ASYNCH: use asynchronous i/o with file system cache
- DIRECTIO: use synchronous i/o with direct i/o
- SETALL: use asynchronous i/o with direct i/o
- NONE: use synchronous i/o with direct i/o

Since the DIRECTIO and NONE options disable the use of asynchronous I/O, they should not be used.

To combine this parameter with the set of the new AIX mount options (DIO & CIO) you should follow following set of the recommendations:

- to enable the filesystem caching for JFS/JFS2 (tends to benefit heavily sequential workloads with low write content), use the default filesystem mount options and set Oracle FILESYSTEMIO_OPTIONS=ASYNCH

- to disable filesystem caching for JFS/JFS2 (DIO tends to benefit heavily random access workloads and CIO tends to benefit heavily update workloads), please follow attached set of the recommendations:

	Oracle 9i	Oracle 10g
JFS	Use “dio” mount options	Set FILESYSTEMIO_OPTIONS=SETALL -or- Use “dio” mount options
JFS2	Use “cio” mount options	Set FILESYSTEMIO_OPTIONS=SETALL -or- Use “cio” mount options

- **DB_WRITER_PROCESS and DBWR_IO_SLAVES**

These parameters specify how many database writer processes are used to update the database disks when disk block buffers in database buffer cache are modified. Multiple database writers are often used to get around the lack of asynchronous I/O capabilities in some operating systems, although it still works with operating systems that fully support asynchronous I/O, such as AIX.

Normally, the default values for these parameters are acceptable and should only be overridden in order to address very specific performance issues and/or at the recommendation of Oracle Support.

- **SHARED_POOL_SIZE**

An appropriate value for the SHARED_POOL_SIZE parameter is very hard to determine before statistics are gathered about the actual use of the shared pool. The good news is that from the Oracle 9i, it is dynamic, and the upper limit of shared pool size is controlled by the SGA_MAX_SIZE parameter. So, if you set the SHARED_POOL_SIZE for an initial value and if you determine that this value is too low, you can change it to a higher one, up to the limit of SGA_MAX_SIZE. Remember that the shared pool includes the data dictionary cache (the tables about the tables and indexes), the library cache (the SQL statements and execution plans), and also the session data if the multi-threaded server (MTS) is used. Thus, it is not difficult to run out of space. Its size can vary from a few MB to very large, such as 20 GB or more, depending on the applications' use of SQL statements. Many application vendors will give guidelines on the minimum size. It depends mostly on the number of tables in the databases; the data dictionary will be larger for a lot of tables and the number of the different SQL statements that are active or used regularly.

- **SGA_MAX_SIZE**

Starting with Oracle 9i, the Oracle SGA size can be dynamically changed. It means the DBA just needs to set the maximum amount of memory available to Oracle (SGA_MAX_SIZE) and the initial values of the different pools: DB_CACHE_SIZE, SHARED_POOL_SIZE, LARGE_POOL_SIZE etc… The size of these individual pools can then be increased or decreased dynamically using the ALTER SYSTEM command, provided the total amount of memory used by the pools does not exceed SGA_MAX_SIZE.

- **SGA_TARGET**

SGA_TARGET specifies the total size of all SGA components. If SGA_TARGET is specified, then the following memory pools are automatically sized:

- Buffer cache (DB_CACHE_SIZE)
- Shared pool (SHARED_POOL_SIZE)
- Large pool (LARGE_POOL_SIZE)
- Java pool (JAVA_POOL_SIZE)
- Streams pool (STREAMS_POOL_SIZE)

If these automatically tuned memory pools are set to non-zero values, then those values are used as minimum levels by Automatic Shared Memory Management. You would set minimum values if an application component needs a minimum amount of memory to function properly.

The following pools are manually sized components and are not affected by Automatic Shared Memory Management:

- Log buffer
- Other buffer caches, such as KEEP, RECYCLE, and other block sizes
- Fixed SGA and other internal allocations

The memory allocated to these pools is deducted from the total available for SGA_TARGET when Automatic Shared Memory Management computes the values of the automatically tuned memory pools.

- **PGA_AGGREGATE_TARGET**

This parameter sets the size of memory (in MB) used to do in-memory sorts. In OLTP environments, sorting is not common or does not involve large numbers of rows. In Batch and DSS workloads, this is a major task on the system, and larger in-memory sort areas are needed. Unlike the other parameters, this space is allocated in the PGA. With Oracle 9's new self manageability features, you do not need to set this parameter as long as your database server is running the dedicated server option and has the WORKING_SET_POLICY to automatic.

The "PGA Memory Advisory " section of the Oracle Statspack or Workload Repository (AWR) report provides statistics that may be useful in determining the optimal PGA Aggregate Target value.

Appendix A: Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this white paper.

1. **Publications**

- "Diagnosing Oracle Database Performance on AIX Using IBM NMON and Oracle Statspack Reports", Dale Martin (IBM)
  - http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100880
- "Configuring IBM TotalStorage for Oracle OLTP Applications", Dale Martin (IBM)
  - http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100319
- "Configuring IBM System Storage DS4000 Series for Oracle Database Applications", Jeff Mucher (IBM)
  - http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100780
- "Tuning SAP R/3 with Oracle on pSeries", Mark Gordon (IBM)
  - http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100377
- "Oracle 9i and 10g Install Guide, Administrator's Reference and Release Notes", Oracle
- "AIX 5L 5.2 & 5.3 Performance Management Guide", IBM
- "IBM AIX Version 6.1 Difference Guide", IBM Redbooks
  - http://w3.itso.ibm.com/abstracts/sg247559.html?Open
- "Dynamic Reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers", Jann-Browning-Burugula (IBM)
- "Improving DB performance with AIX Concurrent I/O": Kashyap-Olszewski-Hendrickson (IBM)

2. **Online Ressources**

- IBM System p and AIX Information Center
  http://publib.boulder.ibm.com/infocenter/pseries/index.jsp

- Oracle Metalink web site:
  http://metalink.oracle.com/

- Oracle 9i Documentation website:
  http://www.oracle.com/technology/documentation/oracle9i.html

- Oracle 10g Documentation website:
  http://www.oracle.com/pls/db102/portal.portal_db?selected=1

- Oracle 11g Documentation website:
  http://www.oracle.com/technology/documentation/database.html