# Tuning IBM AIX 5L for an Oracle Database

*Dennis Massanari and Michel Riviere*
*IBM eServer Solutions Enablement*
*November 2005*

## Table of contents

# Abstract

*This paper discusses performance analysis and tuning for different types of Oracle® workloads and IBM® AIX 5L™ setup configurations. The reader will learn how best to exploit the specific performance features of the AIX® 5L Version 5.3 operating system on IBM eServer™ p5 and pSeries® systems when running an Oracle Database. Tuning techniques include considerations of simultaneous multithreading (SMT) and IBM Micro-Partitioning™ technologies. Discussions of many performance-tuning tools and configuration options, including how to control memory usage, process execution, and configure I/O follow.*

# Introduction

In this paper, the required operating system tuning tasks are grouped into three sections: processor, memory, and I/O tuning. Fine-grained tuning is important and valuable, though this level of tuning can potentially lead to problems if not used correctly. Each of these will be discusses in detail. The reader will come to understand that performance tuning involves trade-offs that might favor one environment, though hurting another. For this reason, the paper also explains the importance of applying tuning changes one at a time, while monitoring the overall system throughput to determine if the alteration is beneficial for the environment.

This paper does not cover performance tuning of the Oracle database related to SQL code tuning, data modeling, or any other aspect of the configuration of the Oracle database itself. Oracle Database generic tuning concepts are widely available and are usually system and hardware independent.

To best exploit the AIX 5L operating system, it will need to be tuned for the specific characteristics of the Oracle application. This is required to achieve the maximumize performance and fully utilize the resources available on the server.

**Note:** Oracle Database generic tuning concepts are widely available and are usually system and hardware independent. These concepts will not be discussed in this paper.

# Tuning processor performance

To gain the most out of the IBM POWER5™ processor, in an eServer p5 system running the AIX 5L operating system, tune the environment to take advantage of Simultaneous multithreading (SMT) and Micro-Partitioning technologies.

## Simultaneous multithreading

SMT is a new feature available on POWER5 systems running AIX 5L V5.3 or IBM i5/OS® Version 5, Release 3. (**Note:** This paper will only address SMT characteristics in the AIX environment.) SMT allows two instruction paths to share access to the POWER5 execution units on every clock cycle. Each instruction path is abstracted as a virtual processor (also referenced as SMT thread of execution), by the operating system, and appears as two logical POWER5 processors, one for each instruction path. For example, a server with four physical processors will have eight logical processors as far as the operating system or Oracle Database is concerned.

SMT improves efficiency by using underutilized hardware resources. The POWER5 processor has two unique contexts with replication of many resources such as registers and program counters. Instructions are fetched and dispatched based on resources available at the time of operation allowing interleaving of instruction streams giving rise to concurrency of instruction execution. For example, one SMT thread might be executing an integer instruction operation while the other might be executing a load point instruction.

One of the key features of SMT technology is that no modification or tuning of the applications is required to benefit from SMT.

SMT is available on POWER5 hardware and can be dynamically enabled or disabled. Enabling or disabling SMT is specific to a partition and allows multiple concurrent partitions to run in different modes (for example, single threading and SMT). To control the modes of SMT, AIX 5L V5.3 provides the *smtctl* command with the following syntax:

`smtcl [ –m off | on [–w boot –w now] ],` where:

- **-m off** : This option will disable SMT mode.
- **-m on** : This option will enable SMT mode.
- **-w boot**: This option makes the SMT mode change effective on next and subsequent reboots.
- **-w now** : This option makes the SMT mode change immediately but will not persist across reboot.

  **Note**: If neither the **-w boot** nor the **-w now** options are specified, then the mode change is made immediately and will persist across subsequent boots.

When SMT is enabled, performance varies from workload to workload. Nevertheless, it is not unusual to observe a significant increase at both the application and system level. Therefore, it is recommended that SMT be enabled by default if the system allows it.

## Process scheduling and priority

On the AIX 5L operating system, the default scheduling policy is fair round robin, also referenced as SCHED_OTHER. This policy implements a classic priority queue round-robin algorithm with one major difference: the priority is no longer fixed. This means, for example, if a task is using a large amount of processor time, its priority level is slowly downgraded to give other jobs an opportunity to gain access to the processor. The drawback of this approach is that sometimes a process can end up with a priority level so low that it does not have an opportunity to run and finish.

Two scheduling commands can change the priority of a process: *nice* and *renice*. By default, a process receives a base priority of 40 and a default nice value of 20. Combined, these two values define the default priority of a process (for example, 60). However, a process can carry a priority ranging between 0 to 255, where priority 0 is the highest and 255 is the lowest (least favorable for the purposes of gaining access to a processor).

The *nice* command is used to start a job with a specific nice value and *renice* is used to apply a new nice value to a process that has already been started. The value of the *nice* and *renice* operand is added to the default 20 base nice value. In other words, a positive value reduces the priority and a negative value increments the priority (the process is more favored to be scheduled). Only the root user can specify a negative nice value with the *nice* and *renice* command.

The easiest way to retrieve the nice value of a process is with the *ps* command and the –l option.

```
        F S  UID      PID     PPID   C PRI NI ADDR     SZ    WCHAN TTY  TIME CMD
   240001 A  1001   442562  323618   0  60 20 59286400  1268  pts/17  0:00 bash
   200001 A  1001  1081422  442562   1  60 20 1f980400   976  pts/17  0:00 ps
```

When a process has a priority so low that it does not have an opportunity to run, its priority must be upgraded to a higher level so that it can finish. This scenario does not happen in regular circumstances, because the process priority will decrease when it is not scheduled for a long period of time.

It is recommended that the priority of Oracle processes not be changed. If the Oracle Database shares the system with other processes that consume too many processor cycles or that need to increase their response times, then the *nice* or *renice* command can be used to impact performance.

Additional AIX 5L tuning capability is available using the *schedtune* or *schedo* command. (For more information on this topic, please consult the *AIX 5L Performance Tools Handbook* found in the **Resources** section of this paper.) These commands must be used with extreme care because of the potentially dangerous effect they might have on the system; they can make the system hang or stop.

## Processor affinity

On a symmetric multiprocessor (SMP) machine, threads are usually scheduled to run on the next available processor. If possible, they run on the same processor. This increases performance because the processor level 1 and level 2 caches already contain code and data for that process (resulting in fewer calls to load from main memory). Improving the scheduling algorithm in this case increases the chances of running on the same processor.

Another alternative is to force the process to run on a particular processor. The benefit is usually an increase of cache versus memory hits. The main down side is that the process then cannot be dispatched to an unused processor. If the processor to which it is allocated is running another process at the same priority, then the process cannot be allocated to another processor (as is normally the case).

Binding a process to a processor is not exclusive. The processor is free to execute other processes. However, all child processes of a bound process inherit the binding constraint. Only the root user can specify a particular binding for a process.

The *bindprocessor* command can be used to bind the main Oracle processes to different processors. (Do not bind database writers and log writers to a processor).

If the SQL*Net listener is bound, all the processes it creates (for example, the Oracle servers dedicated to handle connections from remote application processes) are also bound to that processor. This can be used to limit remote connections to particular processors.

It is easy to start multiple listeners, each listening on its own port, by customizing the $ORACLE_HOME/network/admin/listener.ora file to have multiple listeners defined (the corresponding tnsname.ora file on the client side will also need to be modified).

Once the listeners are started, the following example illustrates how to attach each listener process to a specific processor (the *bindprocessor –q* command displays the processor available):

```
# lsnrctl start listener1
# lsnrctl start listener2
```

```
#
# ps -ef | grep tnslsnr
  oracle  401600 1 0 Jan 15 -  0:00 /home/oracle/bin/tnslsnr
  oracle  696408 1 0 Jan 15 -  0:00 /home/oracle/bin/tnslsnr
#
# bindprocessor 401600 0
# bindprocessor 696408 1
```

In the preceding example, the first listener is attached to processor 0 and the second listener is attached to processor 1.

Processor binding is more difficult when the clients and the Oracle servers run on the same computer using the two-task pipe driver. Determine the process ID for each server process and manually bind it to a processor. The administrative overhead is excessive and probably not worth the effort unless the servers have long process lives.

Since AIX 5L V5.2, the Resource Set API can also be used to ensure the binding of a process on a particular processor or even a particular set of processors. A resource set is a bundle of system resources that can consist of one or more processors, one or more memory pools, or a mixture of memory pools and processors. The *lsrset*, *mkrset*, and *rmrset* commands can be issued to list, create, and delete resource sets, respectively.

In the following example, the *mkrset* command is invoked to create a resource set named *oracle* in the namespace *test* with one processor in it. This processor has the logical processor number 0.

```
# mkrset -c 0 test/oracle
1480-353 rset test/oracle created.
```

Display the new resource set listed together with the default resource sets in the system global repository using the *lsrset* command:

```
# lsrset -a
sys/sys0
sys/node.01.00000
sys/node.02.00000
sys/node.03.00000
sys/node.04.00000
sys/node.05.00000
sys/node.05.00001
sys/mem.00000
sys/cpu.00000
sys/cpu.00001
test/oracle
```

The configuration of the newly created resource set can also be viewed with the *lsrset* command.
The output shows the resource set with logical CPU 0:

```
# lsrset -vr test/oracle
T Name Owner Group Mode CPU Memory
a test/oracle root system rwr-r- 1 0
CPU: 0
MEM: <empty>
```

The *mkrset* command cannot be called to modify an existing resource set. The resource set must be deleted first.

After a resource set is created, the *execrset* command can be used to spawn the Oracle listener in a given resource set. The *execrset* command can also use the default system resource sets for processors and memory. The following examples illustrate both cases. (It is assumed that the same configuration used for the *bindprocessor* example is available).

Start and attach a new listener to a predefined resource set:

```
# execrset test/oracle –e lsnrctl start listener1
```

Start and attach a new listener to a specific processor or set of processors (using the default memory region):

```
# execrset –c 0-3 –e lsnrctl start listener1
```

The user starting the *execrset* command must either have root authority or have CAP_NUMA_ATTACH capability. Unlike the *bindprocessor* command, the *execrset* command cannot be used to dynamically attach a running process to a resource set. The *attachrset* command can be used for this purpose.

To learn more about configuring and using resource sets, refer to *The Complete Partitioning Guide for IBM eServer pSeries Servers* Redbook found in the **Resources** section of this paper.

Processor binding must be handled with care. Processes bound to a processor cannot migrate to different processors even if these processors are free. This might degrade application performance. An environment of homogenous applications with a balanced load is more suitable for processor binding. In addition, many improvements have been made regarding processor affinity to improve performance and workload management has been introduced. Therefore, the *bindprocessor* command is hardly of benefit any more in the AIX 5L operating system.

## Logical partitioning and Micro-Partitioning

Logical partitioning (LPAR) is a feature available on pSeries systems running the AIX 5L operating system. This capability was introduced to create a partition of a computer's processors, memory, and hardware resources environment so that each partition can be operated not only independently but also with its own operating system and applications.

Dynamic logical partitioning (dynamic LPAR) extends the capability of LPAR by providing the ability to attach and detach a managed system's resources logically to and from a logical hardware partition's operating system without rebooting. Processor, memory, and I/O adapters can be released into a free pool that has been acquired from the same pool, or moved to another partition dynamically.

From a performance perspective, LPAR and dynamic LPAR are hardware partitioning with no overhead. However, it is a good practice to try to create an LPAR or dynamic LPAR partition based on processors belonging to the same dual-chip module (DCM) or multichip module (MCM) because it can increase the data locality of the Oracle data in shared cache and minimize the memory access latency. For more information on affinity and migration statistics, consult the IBM Redbooks™ listed in the **Resources** section of this paper (*AIX 5L Differences Guide Version 5.3 Edition* and *Complete Partitioning Guide for IBM eServer pSeries Servers*) regarding LPAR and dynamic LPAR configuration with MCM affinity.

The Micro-Partitioning feature is part of the advanced IBM Virtualization Engine™ technologies and is only available on POWER5 systems. This feature allows multiple partitions to share the processing power of a set of physical processors. A partition can be assigned as little as 1/10th of a physical processor's resource. In most cases, a shared processor pool containing multiple physical processors is shared

among the partitions, and a partition can be configured to cede its processor resource for a short period of time when idle. In this way, spare capacity can be reallocated among partitions (with a granularity of 1/100$^{th}$ of a processor, within 10 milliseconds). This can be beneficial for the performance of a big high priority workload sharing a server with small lower priority jobs running in different partitions.

Micro-Partitioning configuration and tuning, as well as related Virtualization features (such as Virtual I/O and Virtual LAN), can be complex and is covered in the *Advanced POWER Virtualization on IBM eServer p5 Servers: Introduction and Basic Configuration* and *Advanced POWER Virtualization on IBM eServer p5 Servers Architecture and Performance Considerations* Redbooks listed in the **Resources** section of this paper.

The time involved in the management of a large number of partitions can be substantial and might lead to a noticeable degradation of the overall performance of an Oracle Database workload. Also, when the Oracle Database runs in a partition, the number of processors it encounters corresponds to the number of virtual processors available (multiply by two if SMT is available and enabled). This can cause a performance problem if the processing capacity associated with each of these virtual processors in the partition is too low. For example, if a partition is configured with 10 virtual processors, yet needs to use the capacity of only one physical processor, there will be a performance problem.

For these reasons, it is recommended that dedicated partitions (LPAR or dynamic LPAR) be assigned for critical workloads. When using partitions, ensure that enough physical processor capacity is configured. The reason for this is that some operations performed by Oracle are scaled based on the number of physical processors available.

Oracle8i and Oracle9i are not dynamic LPAR-aware and do not dynamically adapt to changes in the system configuration. However, Oracle Database 10*g* is capable of detecting a change in the number of processors dynamically added to, or removed from, a dynamic LPAR, and will change the CPU_COUNT variable accordingly. All subsequent operations can take advantage of the additional processor resources, after increasing the number of processors.

## Tools to monitor processor performance

The AIX 5L environment provides different commands to help investigate and solve a problem in which the system is processor bound or suffers from abnormal processor usage. These commands include: *vmstat*, *iostat*, and *sar*.

### The vmstat command

The first tool is the *vmstat* command, which quickly provides compact information about various system resources and their related performance problems. The *vmstat* command reports statistics about kernel threads in the run and wait queue, memory, paging, disks, interrupts, system calls, context switches, and processor activity.

The reported processor activity is a percentage breakdown of user mode, system mode, idle time, and waits for disk I/O. A new vmstat command option *-I* displays the threads that are waiting on raw I/O to complete, as well as the number of file pages paged in and out per second.

The following example shows the output of the *vmstat -I 2* command:

```
# vmstat -I 2

  kthr      memory                 page                   faults        cpu
-------- ----------- ------------------------ ------------ -----------
r  b  p   avm    fre  fi  fo  pi  po  fr sr  in   sy   cs   us sy id wa
1  0  0 1160603 235   4   8   0   0   0  0 351 1032 1034 23  8 68  1
0  0  0 1160599 235   0   0   0   0   0  0 275 1436 674   6  3 91  0
0  0  0 1160618 235   0   6   0   0   0  0 337 1091 739   4  3 93  0
0  0  0 1160599 235   0   6   0   0   0  0 285  287 644   1  2 97  0
0  0  0 1160602 235   0   0   0   0   0  0 310  554 689   2  8 90  0
0  0  0 1160627 235   0   7   0   0   0  0 341  561 748   2  3 94  1
```

Descriptions of the relevant columns are as follows:

- **kthr**: This set of columns represents kernel thread state changes per second over the sampling interval.
  - **r**: This represents the average number of kernel threads runable over the sampling period, which includes threads already running and threads waiting to run in the run queue. This will be zero on an idle or a lightly loaded system. The higher the number, the busier the processor. A number that is consistently higher than the number of processors in the system indicates a processor performance problem.
  - **b**: This represents the average number of kernel threads in the Virtual Memory Manager (VMM) wait queue. For example, threads might be waiting for journaled file system or enhanced journaled file system (JFS/JFS2) I/O.
  - **p**: This represents the number of threads waiting on raw I/Os (bypassing JFS/JFS2) to complete.
- **page**: This set of columns represents information about page faults and paging activity. These are averaged over the interval and given in units per second.
  - **fi and fo**: These columns represent the number of file pages that have been paged in and out per second.
- **cpu**: This set of columns represents a breakdown of processor time percentage. For multiprocessor systems, processor values are global averages among all processors. Also, the I/O wait state is defined system-wide and not per processor.
  - **us**: This represents the average percentage of processor time executing in user mode.
  - **sy**: This represents the average percentage of processor time executing in system mode.
  - **id**: This represents the average percentage of time that processors were idle and the system did not have an outstanding disk I/O request.
  - **wa**: This represents the average percentage of processor time the processors were idle while the system had an outstanding disk I/O request. This value might be inflated if the actual number of I/O requesting threads is less than the number of idling processors.

The total of the last four columns (us, sys, id, and wa) must equal to 100%, or very close. If the sum of user and system (us and sy) processor-utilization percentages consistently approaches 100%, the system might be encountering a processor performance problem.

## The iostat command

The iostat command is used primarily to monitor I/O devices but also provides processor utilization.

```
#iostat 2

tty:      tin           tout   avg-cpu:  % user   % sys   % idle   % iowait
          0.0           20.9             0.6    10.3    88.8      0.3

Disks:          % tm_act      Kbps      tps    Kb_read   Kb_wrtn
hdisk0            0.5         4.1       1.0         0         8
hdisk1            0.0         0.0       0.0         0         0
```

As with the vmstat command, iostat can display the breakdown of the processor usage. The four columns of the average processor usage must also total around 100%. If the total sum consistently approaches 100%, the system might experience a processor performance problem.

## The sar command

The *sar* command generates two kinds of reports. The first report type displays global system statistics over time. The second type displays process statistics. The sar command can provide queue and processor statistics just as vmstat and iostat do. However, it has two additional features:

- Each sample has a leading time stamp. Thus, an overall average appears after the samples.
- The -P option generates per-processor statistics and global averages for all processors.

The following example shows a typical report from a four-way pSeries system with SMT enabled and per-processor statistics (two samples with one-second intervals are taken in this case).

```
 #sar -P ALL 1 2

AIX stsf 3 5 00CD242F4C00    9/16/04

System configuration: lcpu=4

19:50:18 cpu     %usr     %sys     %wio    %idle    physc
19:50:19  0        1        2        1       96     0.59
          1        0        0        0      100     0.35
          2        9       19        0       72     0.70
          3        0        0        0      100     0.31
          -        4        7        0       89     1.95
19:50:20  0        4        3        0       92     0.66
          1        0        0        0      100     0.35
          2       22       15        1       62     0.72
          3        0        0        0      100     0.28
          -        9        7        0       83     2.00

Average   0        3        3        0       94     0.62
          1        0        0        0      100     0.35
          2       15       17        0       67     0.71
          3        0        0        0      100     0.29
          -        7        7        0       86     1.98
```

In this example, the sar command reported the logical processor statistics, not physical processor statistics, because SMT was enabled.

The sar command does not report the cumulative activity since the last system boot.

# Tuning memory

This section addresses the Oracle process memory footprint, pinning memory, large page memory, as well as tools to monitor memory usage.

## Oracle process memory footprint

The AIX operating system provides a robust environment for running processes with a wide variety of execution characteristics. By taking advantage of the many variables and features to tune the process environment specifically for the Oracle application, memory usage of each Oracle process is reduced.

The AIXTHREAD_SCOPE environment variable can be used for control if a process runs with process-wide contention scope (the default) or with system-wide contention scope. When using system-wide contention scope, there is a one-to-one mapping between the user thread and a kernel thread. On UNIX® systems, Oracle applications are primarily multiprocess and single-threaded. One of the mechanisms that enables this multiprocess system to operate effectively is the AIX post/wait mechanism: **thread_post()**, **thread_post_many()**, and **thread_wait()**. It operates most effectively with the Oracle application when using system-wide contention scope (**AIXTHREAD_SCOPE=S**). As of AIX V5.2, system-wide contention scope will also significantly reduce the memory required for each Oracle process. For these reasons, always export AIXTHREAD_SCOPE=S before starting all Oracle processes. In Oracle Database 10*g* a newer, system-wide contention scope is set internally when the instance and user processes are started.

In AIX V5.1, memory reduction can be achieved by exporting NUM_SPAREVP=1 before starting all Oracle processes. It is best to export AIXTHREAD_SCOPE=S before starting all Oracle processes.

Process memory is divided into different groups based on how it is accessed. The **.text** section contains the program text and read-only (constant) data. The **.data** section contains read/write data. Because the *.text* section is read-only, it can be shared across all the processes in the system executing the same binary. The writeable *.data* has to be unique to each process. Prior to AIX V5.2 with APAR IY50551, constant data that contained a function pointer was placed in the *.data* section. This occurred after load time though the data did not change, including the function pointer. Enhancements were made with APAR IY50551 that allow constant data containing pointers to statically-linked functions to be placed in the *.text* section. This can reduce the memory required by each Oracle process by approximately one megabyte. Oracle applications provide a script and procedure for relinking the Oracle binary to take advantage of this new feature. (Please refer to Oracle MetaLink note 259983.1 provided through the Oracle MetaLink Web site listed in the **Resources** section of this paper.)

## Pinning memory

The AIX operating system allows shared memory to be allocated in pinned memory for improved performance. Pinned memory is always present in real memory; it is not paged in and out. The primary advantage of using pinned memory for the Oracle System Global Area (SGA) is that I/O from pinned memory is optimized for improved path length. This is possible because the asynchronous I/O routines do not need to pin the SGA memory each time it is used for an I/O operation. Because of this performance improvement, it is best to use pinned memory to allocate the Oracle SGA.

When the Oracle initialization parameter LOCK_SGA is set to TRUE, Oracle will allocate shared memory for the SGA using the AIX pinned memory. (**Note:** shmget() is called with the SHM_PIN flag set.) In

addition to setting the Oracle LOCK_SGA parameter to TRUE, the AIX operating system must be configured to allow pinned memory to be used for shared memory. It is also recommended that the maximum amount of allowable pinned memory be set.

In addition to providing pinned memory to applications such as Oracle, which might request it, the AIX operating system requires pinned memory for its own use. It is important to define enough pinned memory for both the Oracle SGA and the AIX operating system. Not providing sufficient pinned memory will cause errors, especially in I/O operations. As an initial guideline, when using pinned memory for the Oracle SGA, set the size of pinned memory to the Oracle SGA size, plus 10% of the real memory on the system if this number exceeds the default, which is 80% of the real memory. Then monitor the availability of pinned memory over time using the *svmon* command during the operation of the application to verify that free pinned memory remains available, and then adjust the amount of pinned memory up or down accordingly, if required.

### Example configuration for using pinned SGA

Assume that real memory equals 128 gigabytes and Oracle SGA equals 110 gigabytes. Then:

Pinned memory = 110 gigabytes + 10% of 128 gigabytes = 122.8 gigabytes or 96% of real memory.

Because 96% is larger than the default for the maximum amount of pinned memory (maxpin%) of 80%, increase this value with the vmo command. If this value was less than the default of 80% then no change is needed.

Use the following commands, as root, to set the pinned memory options. The –p option on the *vmo* commands makes these options persistent across reboots.

- To enable pinning of shared memory segments: # vmo –p –o v_pinshm
- To set the amount of pinned memory available: # vmo –p –o maxpin%=96

## Large page memory

Beginning with the AIX V5.1 operating system when running on IBM POWER4™ or POWER5 processors, pSeries systems support two virtual page sizes: 4 kilobytes (standard page) and 16 megabytes (large page). When using large pages to map virtual memory, the translation look aside buffer (TLB) is able to map more virtual memory with a given number of entries, resulting in a lower TLB miss rate for applications that use a large amount of virtual memory. Additionally, when using large pages, there are fewer page boundaries, which improve the performance of prefetching.

Both online transaction processing (OLTP) and data warehouse environments can benefit from using large pages.

For applications such as Oracle, which typically exploits a large amount of virtual memory, using large page memory will generally result in improved performance. There are three types of memory with which Oracle can use large pages: shared memory (SGA), process data (**.data**), and the instruction text (**.text**). In most Oracle applications, the SGA dominates the virtual memory usage and, consequently, most of the benefit of using large pages is achieved by using them for the SGA. In some special Oracle applications, using large pages for *.data* and *.text* can provide some additional performance benefit. However, because of the granularity of allocation with large pages, each Oracle process might have a larger memory footprint. Large page text and data is supported on AIX V5.3 and only for 64-bit processes and the 64-bit

AIX kernel. For most applications, the recommendation is to use large pages only for the SGA, and not for *.text* or *.data*.

In Oracle 9i and Oracle Database 10*g*, when the Oracle initialization parameter LOCK_SGA is set to TRUE, Oracle will request large pages when allocating shared memory (shmget() call has SHM_LGPAGE flag set). For the AIX operating system to use large pages when allocating shared memory, the Oracle user ID must have CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities. Also, the AIX large page pool must be configured (as shown in the example below). When using large pages on an Oracle Real Application Cluster (RAC) database, where the srvctl command is used to start and stop the RAC database instances, it is also necessary to set the CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities for the root user ID.

Large pages are always pinned, and large page memory cannot be used for standard memory. If a pool of large page memory is configured, then this memory will be unusable for allocation of standard memory even if no other application is currently using large pages.

### Example configuration for using large pages for the Oracle SGA

Give the Oracle user ID the CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities by following these steps:

1.  First check the current capabilities: `#lsuser –a capabilities oracle`

    **Note**: only the root user can display the capabilities attribute.

2.  Add the CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities to the list of capabilities already assigned to this user ID, if any:
    `#chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGAE oracle`
3.  Configure the AIX large page pool by calculating the number of large pages required for the SGA:
    `num_of_large_pages = INT((total_SGA_size-1)/16MB)+1`
4.  Configure the number and size of large pages:
    `#vmo -p -o lgpg_regions=num_of_large_pages -o lgpg_size=16777216`

    **Note**: If a reboot is required on the system, because of changing the large page configuration, this will be indicated in the output of the *vmo* command.

### Example configuration for using large pages for .text and .data

These configuration steps are in addition to the steps required for using large page SGA:

1.  Calculate the additional large pages required for **.text** and **.data**.
2.  Start the Oracle instance and execute the following command as root:

```
#svmon –P <PID an Oracle process>
for example:
#svmon -P 552974
-------------------------------------------------------------------------
     Pid Command          Inuse      Pin   Pgsp  Virtual 64-bit Mthrd LPage
  552974 oracle           95447     4012      0    86224      Y     N     N

    Vsid       Esid Type Description          LPage   Inuse   Pin Pgsp Virtual
  181958   70000000 work default shmat/mmap       -   58243     0    0 58243
  1e197e   70000001 work default shmat/mmap       -   12663     0    0 12663
```

```
  1a17da        10 clnt text data BSS heap         -   9212     0    -    -
       0         0 work kernel segment             -   7821  3990    0  7821
  d08ad  90000000 work loader segment              -   4776     0    0  4776
  1f1a7f        11 work text data BSS heap         -   1924     0    0  1924
  121272  90020014 work shared library text        -    375     0    0   375
   21a82  8001000a work private load               -    138     0    0   138
    1a40  9001000a work shared library text        -    113     0    0   113
   d1a6d  80020014 work private load               -    107     0    0   107
              :
              :
```

The row with an Esid of 10 corresponds to the **.text** segment, and the row with an Esid of 11 corresponds to the **.data** segment. When checking the **.text** segment for all the Oracle processes, notice that they all have the same Vsid. This is because they share this read-only segment. The **.data** section is private to each Oracle process so that the total large page requirement is dependant on the number of foreground and background Oracle processes when using large page data. The column labeled **Inuse** contains the number of four kilobyte pages in that segment.

Use the following formula to calculate the number of additional pages for large page **.text** and **.data**:

```
num_of_additional_pages = INT((.text_pages+4095)/4096) + N*(INT(.data_pages+4095)/4096)
```

Using the *svmon* example output and assuming 25 foreground and background processes (N=25), then:

```
num_of_additional_pages = INT((9212+4095)/4096) + 25*(INT((1924+4095)/4096)) = 28
```

Add this number to the large pages needed for the SGA and execute vmo:

```
#vmo -p -o lgpg_regions=num_of_large_pages+num_of_additional_pages -o lgpg_size=16777216
```

Edit the XCOFF file header in the oracle binary to enable it to use large page data:

```
#ldedit –b lpdata
```

Prior to starting the Oracle instance and listener, export LDR_CNTRL in the Oracle user ID used to start the instance and listener:

```
    #export LDR_CNTRL=LARGE_PAGE_TEXT=Y@LARGE_PAGE_DATA=M
```

**Note**: This must be set in addition to editing the XCOFF file header to allow both large page text and large page data. Setting the LARGE_PAGE_DATA=M option allocates only enough large pages for the data segment up to the brk value. The **@** character is used to separate the definition of multiple values to the LDR_CNTRL variable.

## Tools to monitor memory usage

This section discusses tools to monitor memory usage, such as the vmstat and svmon commands.

### vmstat

The *vmstat* command also aids with monitoring memory usage.

Use the *vmstat –l* command to monitor the allocated and free large pages.

```
$ vmstat –l

System configuration: lcpu=2 mem=7424MB
```

```
kthr    memory                  page                    faults         cpu      large-page
----- -----------  ------------------------  ------------  -----------  -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa   alp   flp
 1  1 433591 338295   0   0   0   0    0   0  85  382 340  0  0 99  0     0      0
```

In this vmstat output, the alp column indicates the number of allocated large pages, and the flp column is the number of free large pages.

### svmon

The *svmon* command is useful for looking at the memory used by a process or a group of processes. The *–P <pid>* option used in the **Large page memory** section shows the memory usage for a specific process. The *–C oracle* option shows the memory usage for all of the Oracle processes running on the system. Only root authority can use the *svmon* command.

# Tuning I/O performance

The processor and memory can be tuned to peak efficiency for Oracle, but if the I/O processes are not tuned properly, the Oracle Database can still show performance problems. For example, if the IBM POWER5 processor is set to run SMT, but the I/O subsystem only allows one process at a time, the system will run as though it only has a single-threaded processor. The solution to this problem is to enable asynchronous I/O, which allows several I/O processes to run concurrently. This and other preferred I/O tuning practices are discussed in this section.

## Asynchronous I/O

Asynchronous I/O allows a program to initiate I/O and continue execution of useful work, while other I/O operations are carried out in parallel by the operating system. Because Oracle applications often require multiple server and user processes at the same time, they take advantage of asynchronous I/O to overlap program execution with I/O operations. Asynchronous I/O is used with Oracle on the AIX operating system to improve system performance.

In AIX V5.2 and newer versions, two asynchronous I/O subsystems are supported, the original asynchronous I/O (called Legacy asynchronous I/O), and the new POSIX asynchronous I/O. The two types of asynchronous I/O differ in the way the asynchronous I/O subsystem API is defined. Their performance characteristics are the same. All Oracle releases up to and including the current release, Oracle Database 10*g*, use Legacy asynchronous I/O.

### Asynchronous I/O servers

In the AIX operating system, kernel processes are used for asynchronous I/O to a file system (JFS, JFS2, or General Parallel File System [GPFS]). Referred to as asynchronous I/O servers, these kernel processes handle each asynchronous I/O request. When performing asynchronous I/O to raw devices with the asynchronous I/O fast path enabled, asynchronous I/O servers are not used. Therefore, the following discussion on tuning minservers and maxservers does not apply. The asynchronous I/O fast path is enabled by default.

When doing asynchronous I/O to a file system, each asynchronous I/O operation is tied to an asynchronous I/O server. Thus, the number of asynchronous I/O servers limits the number of concurrent asynchronous I/O operations in the system. The initial number of servers started at boot time is determined by the minservers parameter, which has a default value of one. As more concurrent asynchronous I/O operations occur, additional asynchronous I/O servers are started, up to the maxservers value, which has a default of 10. If using Oracle with data files on a file system, the default values for minservers and maxservers are too small and need to be increased. Starting with the AIX V5.2, these parameters are implemented on each processor. Prior to AIX V5.2, these values were system-wide limits.

If using JFS or JFS2, set the initial maxservers value using the following formula:

| | |
|---|---|
| AIX V5.1 and older: | maxservers=(10 * number of logical disks) |
| AIX V5.2 and newer: | maxservers=(10 * number of logical disks / number of processors) |

To fine-tune the number of asynchronous I/O servers, adjust the initial value of maxservers and monitor the performance effects on the system during periods of high I/O activity. The downside of too many asynchronous I/O servers is the increased memory and processor overhead caused by additional processes.

If using GPFS, set maxservers based on the number of GPFS worker1Threads, according to the following recommendation:

```
AIX V5.1 and older:   maxservers=number of GPFS worker1Threads
AIX V5.2 and newer: maxservers=slightly more than the GPFS worker1Threads/number_of_CPUs
```

For details on tuning maxservers for Oracle and GPFS, refer to the *GPFS for AIX FAQs* found in the **Resources** section of this paper.

There are separate asynchronous I/O servers for Legacy asynchronous I/O and POSIX asynchronous I/O. For Oracle, only the number of Legacy asynchronous I/O servers matters.

Use the following command to determine the current minservers and maxservers settings:

```
#lsattr –El aio0

Sample output

autoconfig  available    STATE to be configured at system restart    True
fastpath    enable       State of fast path                          True
kprocprio   39           Server PRIORITY                             True
maxreqs     40960        MAXIMUM number of REQUESTS                  True
maxservers  256          MAXIMUM number of servers per cpu           True
minservers  20           MINIMUM number of servers                   True
```

To change the minservers and maxservers, use the *smit* system management interface or execute the following command from the root user:

```
#chdev -P -l aio0 -a maxservers='m' -a minservers='n'
```

**Note**: Use the –P option to force the change to remain in effect after a reboot. If the asynchronous I/O subsystem is already in the available state, then the –P option is required, and the system must be rebooted for this change to take effect.

The initialized asynchronous I/O server processes appear in the ps output command with the name aioserver for Legacy asynchronous I/O servers and posix_aioserver for POSIX asynchronous I/O.

AIX 5L V5.3 provides a new –A option to the *iostat* command, which gives useful information for tuning the number of asynchronous I/O servers and maxreqs. The asynchronous I/O report has the following columns:

- **avgc**: The average global asynchronous I/O request count per second for the specified interval.
- **avfc**: The average fastpath request count per second for the specified interval.
- **maxgc**: The maximum global asynchronous I/O request count since the last time this value was fetched.
- **maxfc**: The maximum fastpath request count since the last time this value was fetched.
- **maxreqs**: The maximum asynchronous I/O requests allowed.

Use the avfc and maxfc columns for fastpath asynchronous I/O requests as they do not use an asynchronous I/O server.

### maxreqs

In addition to tuning minservers and maxservers for asynchronous I/O, maxreqs will also need to be tuned. The maxreqs value specifies the maximum number of asynchronous I/O requests that are waiting for processing in the system at one time. This value includes the asynchronous I/O operations that are in progress in asynchronous I/O servers as well as the requests that are queued but not in progress.

If the maxreqs value is too low, then the following Oracle warning message might be returned:

```
Warning: lio_listio returned EAGAIN
```

This warning message indicates a need to increase the maxreqs value. When performing file system I/O, a maxservers value that is too low can also indirectly cause this warning message. A low maxservers value can limit the rate at which asynchronous I/O requests are completed, resulting in an increase in the outstanding I/O requests during a period of heavy I/O activity. The same is true for a performance problem in the physical I/O subsystem. For example, if the data is poorly distributed across the disks, data access can be slow.

## File system and raw I/O

Oracle data files can be located in file systems or raw devices. The AIX operating system has special features to enhance the performance of file system I/O for general-purpose file access. These features include read ahead, write behind, and I/O buffering, which can provide huge performance benefits for many applications. However, Oracle employs its own I/O optimizations and buffering that, in most cases, are redundant to those provided by the AIX file systems. Oracle uses buffer cache management involving data blocks in shared memory. The AIX operating system uses virtual memory management (VMM) involving data buffered in virtual memory. If both separately try to manage data caching, it will result in wasted memory, processor overhead, and suboptimal performance.

It is generally better to allow Oracle to manage I/O buffering because it has information regarding the context in which the data is being referenced, and therefore, it can better optimize memory usage for I/O buffering. Oracle manages buffered data in the buffer cache and disk access based on the type of SQL query accessing the data. Examples of this include Oracle's use of the recycle pool when accessing data it expects not to be re-referenced, and its use of the db_file_multiblock_read_count parameter to increase the read block size when performing large table or index scans.

When the file system I/O buffer cache is used, I/O data is copied into the buffer cache, and then a second copy is written into the application I/O buffer. This causes additional path length for each I/O operation to a file system compared to raw I/O.

For data integrity, Oracle opens all data files with the O_DSYNC flag. When the O_DSYNC flag is used, it guarantees that all the data and metadata required to access the data have been safely written to nonvolatile storage when a write completes. This ensures that the data can be accessed even after a system crash. The O_DSYNC flag also disables the file system write behind algorithms.

For some special database operations, the I/O will bypass the Oracle buffer cache, but can benefit from using the file system I/O buffer cache. These special operations include: reads and writes to the TEMPORARY table space, data in NOCACHE large objects (LOBs), and parallel query slaves reading data.

For optimal performance, use raw devices or raw logical volumes for Oracle data files. Those who desire the flexibility of using file systems can use concurrent I/O with JFS2 to achieve performance that is close to raw. Concurrent I/O is only available with JFS2 and AIX V5.2 Modification Level 01 or newer. Users with application data in JFS file systems can use direct I/O. The differences in these options are described below:

### Raw I/O

When accessing Oracle data files on raw devices, there are fewer AIX parameters to tune than when using data files on file systems. This is because with raw I/O, the file system and VMM layers are bypassed, resulting in less path length per physical I/O operation. If Oracle can efficiently buffer I/O data for the workload, such that the number of physical I/O operations does not increase, this will result in improved performance. When using raw devices, contention on inode locks is also avoided.

When using raw devices with Oracle on the AIX operating system, the devices are raw logical volumes or raw disks. When using raw disks for I/O, the logical volume manager (LVM) layer is also bypassed. The LVM allows the grouping of disks into volume groups (VG), and then defines logical volumes (LV) where the size and number of LVs is independent of the size and number of disks, within the limitation of the total space in the VG. Because of this capability, the use of raw LVs is recommended for Oracle data files unless the Oracle ASM is used. Oracle ASM includes the capability to create Oracle data files, which do not need to be mapped directly to the size and number of disks. With Oracle ASM, using raw disks is preferred.

When using raw devices with Oracle RAC databases, which require shared access to the data from all instances, the devices must be raw logical volumes in concurrent volume groups or raw shared disks. The use of concurrent volume groups requires High-Availability Cluster Multi-Processing for the AIX operating system (IBM HACMP® for AIX). For configurations without HACMP clustering, which requires shared disks, use Oracle ASM to manage the mapping of data files to the disk group.

When accessing raw fiber channel disks (without the LVM), install the authorized program analysis report (APAR) IY57148 or upgrade to AIX 5L V5.2 Modification Level 04 or newer (including AIX 5L V5.3). This APAR provides locking improvements in the fiber-channel device driver to improve raw performance.

### pbufs

When accessing LVs in a buffer in pinned memory, pbuf is used to contain information associated with each pending disk I/O request. If an I/O request is made to an LV and there are no free pbufs, then the I/O request will be delayed. Use the following command, from the root user, to determine blocked I/Os:

| | |
|---|---|
| AIX V5.1 and older: | /usr/samples/kernel/vmtune –a|grep hd_pendqblked |
| AIX V5.2 and newer: | /usr/bin/vmstat –v|grep "blocked with no pbuf" |

If this command returns a value other than 0, there might be a benefit from increasing the number of pbufs.

Use the following command, from the root user, to determine the current number of pbufs:

| | |
|---|---|
| AIX V5.1 and older: | /usr/samples/kernel/vmtune –a|grep hd_pbuf_cnt |
| AIX V5.2 and newer: | /usr/sbin/ioo –a|grep pv_min_pbuf |

Use the following command, from the root user, to change the number of pbufs:

| | |
|---|---|
| AIX V5.1 and older: | /usr/samples/kernel/vmtune –B *<number of pbufs>* |
| AIX V5.2 and newer: | /usr/sbin/ioo –p –o pv_min_pbuf=*<number of pbufs per PV>* |

*Note*: The –p flag makes the change stay in effect after a reboot.

The pv_min_pbuf value in AIX V5.2 sets the minimum number of pbufs for each physical volume (PV) that the LVM uses.

AIX 5L V5.3 provides a new command, *lvmo*, which can be used to fine-tune the usage of pbufs. Use the *lvmo* command to tune the number of LVM pbufs on a per volume group basis. The *lvmo* command allows the following parameters:

- **pv_pbuf_count**: The number of pbufs that will be added when a physical volume is added to the volume group.
- **max_vg_pbuf_count**: The maximum number of pbufs that can be allocated for the volume group. For this value to take effect, the volume group must be varied off and varied on again.

Monitor the system performance before and after increasing the pbufs to make sure there are no adverse effects. The pbufs are in pinned memory; thus making them unnecessarily large will reduce the availability of memory for other purposes.

### LTG size

The logical track group size (LTG size) is the maximum transfer size allowed for disk I/O. The LTG size is defined at the VG level; therefore, all disks in a VG will have the same LTG size. For VGs created with the –I option on mkvg, the LTG size is defined with the –L option on mkvg. It must be set no larger than the smallest disk transfer size of all disks in the VG. The default LTG size in this case is 128 kilobytes. In AIX V5.3 and newer versions of the AIX 5L operating system, if the VG is created without the –I flag, the LTG size is automatically set to the smallest disk transfer size of all disks in the VG when the VG is varied on. Prior to the AIX 5L environment, the LTG size was always 128 kilobytes.

The maximum transfer size used by Oracle on the AIX operating system is one megabyte. Oracle will use large transfer sizes when doing large table scans, and the transfer size will be the db_file_multiblock_read_count times the database block size. When the Oracle transfer size is large, there will be a performance benefit from increasing the LTG size. An Oracle application with many large table scans might also see a performance benefit from using a large LTG size.

The maximum transfer size allowed is dependant on the storage system. Monitor the performance before and after changing this parameter to make sure the change does not have a negative impact on the workload.

## The lvm_bufcnt command

The lvm_bufcnt command specifies the number of LVM buffers available for raw I/O. When the application transfer size exceeds the LTG size, the I/O is broken down into a unit of up to lvm_bufcnt I/Os each having a size of LTG size. If the application transfer size exceeds the size of one unit, then the I/O is done in multiple units, and the I/O for each unit must complete before the I/O for the next unit can begin. The default lvm_bufcnt value is 9 and the maximum value is 64. For optimal performance, when the LTG size cannot be increased to the application transfer size because of limitations in the maximum transfer size in the disk, make sure the product of the lvm_bufcnt and the LTG size is greater than or equal to the application transfer size. This ensures that the I/O can complete in one unit.

Use the following command from the root user to display lvm_bufcnt:

| | |
|---|---|
| AIX V5.1 and older: | usr/samples/kernel/vmtune –a|grep lvm_bufcnt |
| AIX V5.2 and newer: | /usr/sbin/ioo –a|grep lvm_bufcnt |

Use the following command from the root user to set lvm_bufcnt.

| | |
|---|---|
| AIX V5.1 and older: | usr/samples/kernel/vmtune –u <number of lvm_bufcnt> |
| AIX V5.2 and newer: | /usr/sbin/ioo –p –o lvm_bufcnt=<number of lvm_bufcnt> |
| | *__Note__: The –p flag makes the change stay in effect after a reboot.* |

## Direct I/O

As described above, some applications such as Oracle do not benefit from file system caching. For these applications, there is a direct I/O option for file systems, which bypasses the file system cache, and the extra data copy associated with using it. The direct I/O option is available on the JFS, JFS2, and GPFS file systems.

Direct I/O requires certain restrictions on the source and target data alignment as well as the data transfer size. If there are multiple opens of a file in the system, and any of these opens are without direct I/O, accesses to this file from all the opens will be demoted to use normal I/O until the last non-direct I/O open is closed. The details of the alignment and size restrictions for using direct I/O are documented in the AIX 5L manual *General Programming Concepts: Writing and Debugging Programs, Working with File I/O* found in the **Resources** section of this paper. Oracle generally does a good job meeting the conditions required to use direct I/O. When the direct I/O restrictions are not met, the I/O operations do not fail; they are demoted by the AIX operating system to nondirect I/O operations.

There are several ways to specify that direct I/O can be used. The *dio* option on the mount command will result in the AIX operating system using direct I/O for all I/O operations to that file system, assuming the previously mentioned conditions are met. The application can specify direct I/O for specific files by using the O_DIRECT flag when the file is opened. When using the *dio* mount option, no application changes are required to take advantage of direct I/O.

Oracle Database 10*g* will open data files located on the JFS file system with the O_DIRECT option if the filesystemio_options initialization parameter is set to either **directIO** or **setall**. The *setall* option can be used because asynchronous I/O is disabled when *directIO* is specified. For all Oracle Database versions supporting the GPFS file system, direct I/O is always used for all data files located on a GPFS file system.

## Concurrent I/O

AIX 5L V5.2 Modification Level 01 and newer versions provide a feature for JFS2 called concurrent I/O. Concurrent I/O includes the performance benefits previously available with direct I/O, plus the additional performance benefit of eliminating contention on the inode lock. Concurrent I/O can be used by applications such as Oracle that do their own data serialization. These applications do not require the functions of the AIX operating system inode lock.

As with direct I/O, concurrent I/O can be specified for the entire file system by specifying the *cio* mount option, or in the application for a particular file by using the O_CIO open flag.

Concurrent I/O provides better performance than direct I/O, but has some additional restrictions. Concurrent I/O is only available for JFS2 file systems. Mixed open modes for a file are not supported with concurrent I/O. If a process attempts to open a file in concurrent I/O mode, and that file is already open by some other process in the system without concurrent I/O, then the concurrent I/O open will fail. If a file is opened in concurrent I/O mode, and another process tries to open that same file without concurrent I/O, then the non-concurrent I/O open will fail. The use of concurrent I/O is intended only for special applications, such as Oracle, which implement their own data serialization. When using the *cio* mount option on a file system that file system can only contain Oracle data and log files. Never use the *cio* mount option on a file system containing shared libraries or executables, such as ORACLE_HOME.

Oracle Database 10*g* will open data files located on the JFS2 file system with the O_CIO option if the filesystemio_options initialization parameter is set to either **directIO** or **setall**. The *setall* option can be used, as asynchronous I/O is disabled when *directIO* is specified. Note that *directIO* is used for both direct I/O and concurrent I/O, and Oracle checks if the data file is on JFS or JFS2 and uses direct I/O or concurrent I/O accordingly.

## Enhanced journal file system (JFS2) and concurrent I/O

For non-RAC database users who prefer the flexibility of using a file system rather than raw devices, the best choice is the enhanced journal file system (JFS2) with concurrent I/O. When properly tuned, JFS2 with concurrent I/O will provide performance close to that of raw devices.

The I/O alignment restrictions that determine when the file systems buffer cache will be bypassed with direct I/O or concurrent I/O are dependent on the JFS2 block size (agblksize). For optimal I/O performance, agblksize needs to be equal to the size of the smallest block size used when accessing

the file system. This is the largest agblksize that will meet the alignment restrictions for bypassing the file system buffer cache. The smallest block size used by Oracle is the database block size. For the redo log files, the smallest block size used is 512 bytes. For this reason, always put the redo log files on a separate file system with an agblksize of 512, and put the data files on one or more file systems with the agblksize equal to the smallest database block size used for table spaces in data files on that file system.

When the database has multiple database block sizes, it might be advantageous to group data files by the database block size specified in the table spaces they contain, and set the agblksize to this value. If a separate file system is not used for each database block size, the agblksize of the file system can be that of the smallest database block size used.

## Journaled file system and direct I/O

When using journaled file system (JFS) file systems, direct I/O is recommended.

Starting with AIX 5L V5.1, a new type of lock is used for inode locking for the JFS file system. The new lock is a read/write lock. It allows multiple readers to get data from a file at the same time without being blocked by an inode lock. In most cases, the new type of lock results in less contention and better performance. However, when doing sequential reads, the old-style lock might give better performance. The ability to dynamically specify the type of inode lock used for JFS was added to AIX 5L V5.1 with APAR IY45136 and is included in AIX V5.2 and above.

To use the old-style exclusive inode lock, use the following command from the root user:

| | |
|---|---|
| AIX V5.1: | # /usr/samples/kernel/vmtune -E 0 |
| AIX V5.2 or newer: | # ioo -p -o jfs_use_read_lock=0 |
| | ***Note**: The –p flag makes the change stay in effect after a reboot.* |

To use the new style read/write lock, use the following command from the root user.

| | |
|---|---|
| AIX V5.1: | # /usr/samples/kernel/vmtune -E 1 |
| AIX V5.2 or newer: | # ioo -p -o jfs_use_read_lock=1 |
| | ***Note**: The –p flag makes the change stay in effect after a reboot.* |

## GPFS

For users who want to use a file system with Oracle RAC databases, GPFS is available. Oracle Database 9i Release 2 and 10*g* support the GPFS file system and automatically open all data files located on GPFS with direct I/O.

For information on tuning GPFS for Oracle, refer to the GPFS for AIX FAQs link located in the **Resources** section of this paper.

## Automated storage management

Oracle Database 10*g* has a new Oracle Automated Storage Management (ASM) component that manages Oracle Database files. ASM is configured by assigning raw devices to ASM disk groups. When a table space is created, the disk group where the table space will be located is specified. A template is also specified, which describes the storage attributes of the data. Using this information, ASM allocates the table spaces with the desired properties. Among other attributes, ASM handles mirroring and striping based on the template. A predefined set of templates provides for different database file types.

When using ASM, the database files are on raw devices. These raw devices are not in the AIX volume groups or logical volumes, and therefore VG and LV AIX tuning parameters do not apply. The other AIX I/O tuning is the same as when using raw devices or raw logical volumes without the ASM.

## Tools to monitor I/O performance

Several commands allow AIX I/O performance monitoring. Some of these commands also allow the monitoring of processor and other system performance. This section explains how to use these commands.

### vmstat

The *vmstat* command contains a useful overview of system performance, including some parameters related to I/O.

Use the *vmstat –I* option to get an I/O oriented view of the system performance:

```
$ vmstat -I
System Configuration: lcpu=23 mem=98304MB
  kthr      memory              page              faults        cpu
-------- ----------- ------------------------ ------------ -----------
 r  b  p   avm    fre  fi  fo  pi  po  fr  sr   in   sy  cs us sy id wa
 1  1  0 9718143 15206735   2   8   0   0   0    0   47  857 293  0  0 99  1
```

The following columns are of special interest to I/O performance:

- **r**:   The average number of kernel threads runnable over the sampling period. This includes threads already running and threads waiting to run in the queue.
- **b**:   The average number of kernel threads in the VMM wait queue (for example, waiting for journaled file system [JFS/JFS2] I/O).
- **p**:   The number of threads waiting on actual physical I/O per second.
- **fi**:   The number of file page-ins per second.
- **fo**:   The number of file page-outs per second.
- **wa**:   CPU idle time during which the system had outstanding disk/NFS I/O request (requests).

A consistently high value in the **r** column indicates that there are processes ready to run, but waiting for processor resources. This indicates a processor rather than an I/O performance problem. Consistently high values of **b** and **wa** indicate that processes are waiting on the file system I/O and processor capacity is available, which indicates that there might be potential for improved performance by tuning the physical or file system I/O. A consistently high value in the **p** and **wa** columns indicates there might be potential for tuning the physical I/O.

### iostat

The primary use of the *iostat* command is to report system I/O statistics. Typically, the command is used with an interval and count value, to show the I/O statistics over a number of fixed intervals.

The following sample *iostat 10 1* output shows the I/O statistics for one 10-second interval:

```
$ iostat 10 1
System configuration: lcpu=2 drives=21
tty:      tin        tout  avg-cpu: % user    % sys     % idle    % iowait
         0.0         4.2                0.9      1.6      49.0      48.5
Disks:       % tm_act      Kbps      tps   Kb_read   Kb_wrtn
dac0             0.0       9.2       3.8        82        10
dac0-utm         0.0       0.0       0.0         0         0
hdisk20          0.0       9.2       3.8        82        10
hdisk21          0.0       0.0       0.0         0         0
hdisk22          0.0       0.0       0.0         0         0
hdisk4           0.0       0.0       0.0         0         0
hdisk8          29.2    6553.6      51.2         0     65536
hdisk9          28.6    6553.6      51.2         0     65536
hdisk10          0.0       0.0       0.0         0         0
hdisk7           0.0       0.0       0.0         0         0
hdisk6          40.3    8704.0      68.0         0     87040
hdisk12          0.0       0.0       0.0         0         0
hdisk11          0.0       0.0       0.0         0         0
hdisk13          0.0       0.0       0.0         0         0
hdisk15          0.0       0.0       0.0         0         0
hdisk5           0.6       3.6       0.9         0        36
hdisk14          0.0       0.0       0.0         0         0
hdisk16          0.0       0.0       0.0         0         0
hdisk18          0.0       0.0       0.0         0         0
hdisk17          0.0       0.0       0.0         0         0
hdisk19          0.0       0.0       0.0         0         0
```

The definitions for the columns in the sample output are as follows:

- **% tm_act**: Indicates the percentage of time the physical disk was active (bandwidth utilization for the drive).
- **Kbps**: Indicates the amount of data transferred (read or written) to the drive in *kilobytes per second*.
- **tps**: Indicates the number of *transfers per second* that were issued to the physical disk. A transfer is an I/O request to the physical disk. Multiple logical requests can be combined into a single I/O request to the disk. A transfer is of indeterminate size.
- **Kb_read**: Indicates the total number of kilobytes read.
- **Kb_wrtn**: Indicates the total number of kilobytes written.

AIX 5L V5.3 provides a new –A option for *iostat* that shows asynchronous I/O statistics. By default, the –A options shows the asynchronous I/O statistics by disk, or the –Q option can be added to show the asynchronous I/O statistics by file system. When using the –A option, the same sets of statistics are shown, but only for asynchronous I/O.

## The filemon command

The *filemon* command reports the I/O activity for each of the layers in the I/O subsystem: logical files, virtual memory segments, logical volumes, and physical volumes. For each layer, *filemon* reports a summary of the I/O characteristics.

The following section of output from *filemon* shows the I/O statistics at the logical volume level for one of the logical volumes used by a file system containing an Oracle data file:

```
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------
VOLUME: /dev/fslv19  description: /j2r5A
reads:                 4       (0 errs)
  read sizes (blks):   avg    16.0 min     16 max      16 sdev     0.0
  read times (msec):   avg 371.190 min  66.367 max 479.576 sdev 176.161
  read sequences:      4
  read seq. lengths:   avg    16.0 min     16 max      16 sdev     0.0
writes:                12923  (0 errs)
  write sizes (blks):  avg    16.0 min     16 max      16 sdev     0.0
  write times (msec):  avg 460.946 min   8.925 max 1390.970 sdev 158.716
  write sequences:     12923
  write seq. lengths:  avg    16.0 min     16 max      16 sdev     0.0
seeks:                 12927  (100.0%)
  seek dist (blks):    init 13318528,
                       avg 937146.4 min     16 max 14107768 sdev 2401696.8
time to next req(msec): avg   2.498 min   0.001 max 3126.228 sdev  34.707
throughput:            3183.6 KB/sec
utilization:           0.82
```

In this sample output, notice that during the *filemon* sample interval there were three reads and 12,923 writes. All of the writes were the same size, 16 blocks, or 8 kilobytes, which was the database block size of the database.

*Tuning IBM AIX 5L for an Oracle Database*

# Summary

Oracle performance tuning is a vital part of the management and administration of successful Oracle Database systems, and requires tuning both the Oracle Database and the operating system to work efficiently together. This paper presents how to tune the AIX 5L operating system (including the AIX 5L Version 5.3 operating system) to achieve the best performance with the Oracle Database, and to leverage some of the new hardware-based features provided by the latest IBM eServer p5 systems with POWER5 processors, including SMT and Micro-Partitioning.

Tuning the processor for greater performance can be done through SMT and scheduling and prioritizing Oracle Database tasks. Processors can be assigned specifically to certain tasks and logical partitioning can be used to improve Oracle throughput. Administrators and developers can also use the vmstat, iostat and sar commands to analyze performance.

The AIX memory subsystems can also be tuned, including the option of reducing the memory footprint of the Oracle processes, using pinned memory, and using large pages to achieve better performance.

Specific performance tuning of the I/O subsystem for different types of Oracle Database deployments can be done, depending on the use of raw devices (logical volumes or disks), file systems, ASM, and GPFS.

In each section of this paper, the most useful tools to configure, monitor, and evaluate the performance of the tuning aspects were covered.

# Resources

These Web sites provide useful references to supplement the information contained in this document:

- IBM System p5 servers
  **ibm.com**/systems/p
- IBM AIX 5L Web site
  **ibm.com/**servers/aix
- GPFS for AIX FAQs and the HACMP library
  publib.boulder.ibm.com/infocenter/clresctr/index.jsp?topic=/com.ibm.cluster.gpfs.doc/gpfs_faqs/
  gpfs_aix_faq.html
- *AIX 5L Performance Management Guide, Using POWER4-based Systems*
  publib16.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixbman/prftungd/prftungd03.htm

**White papers**
- Improving Database Performance With AIX Concurrent I/O
  **ibm.com**/servers/aix/whitepapers/db_perf_aix.pdf

**IBM Redbooks™** (i**bm.com**/redbooks)
- Advanced POWER Virtualization on IBM eServer p5 Servers:
  - o Introduction and Basic Configuration (SG24-7940)
  - o Architecture and Performance Considerations (SG24-5768)
- A Practical Guide for Resource Monitoring and Control (SG24-6615)
- The Complete Partitioning Guide for IBM eServer pSeries Servers (SG24-7039)
- Effective System Management Using the IBM HMC for pSeries (SG24-7038)
- Problem Solving and Troubleshooting in AIX 5L (SG24-5496)
- Understanding IBM eServer pSeries Performance and Sizing (SG24-4810)
- AIX Version 5.3 Differences Guide (SG24-5766)
- AIX 5L Performance Tools Handbook (SG24-6039)

**Techdocs: The technical sales library**
- How to find which AIX versions are certified with Oracle Database releases
  **ibm.com**/support/techdocs/atsmastr.nsf/WebIndex/TD101737

**Oracle Web sites**
- Oracle Database 10*g*
  www.oracle.com/database/index.html
- Oracle Real Application Clusters (RAC) with Oracle Database
  www.oracle.com/database/rac_home.html
- Oracle MetaLink
  metalink.oracle.com
- Oracle Database Performance Tuning Guide, 10g Release 1 (10.1) (B10752-01)
  www.oracle.com/pls/db10g/portal.portal_demo3?selected=1
  - o Oracle Technology Network for current Oracle Database documentation
    www.oracle.com/technology/documentation/index.html

# About the authors

*Dennis Massanari*

IBM eServer Solution Enablement
Dennis Massanari is a senior software programmer in the IBM eServer Solutions Enablement organization, where he assists solution providers in enabling applications for the AIX operating system on the IBM eSeries pSeries platforms. He has 22 years of experience in the area of performance, initially developing performance measurement tools for IBM eServer zSeries® processors. For the past eight years, he has worked on porting, performance, and benchmarks with Oracle Database. He holds a Bachelor of Science in Computer Engineering from the University of Illinois. Dennis can be contacted at massanar@us.ibm.com.


*Michel Riviere*

IBM eServer Solution Enablement
Michel Riviere is a senior software programmer in the IBM eServer Solutions Enablement organization, where he assists solution providers in enabling applications for the AIX and Linux™ operating systems on the pSeries platform. His areas of expertise include performance and benchmarks with Oracle Database, both the compiler and operating system. He holds a PhD in Computer Science from the University Joseph Fourrier at Grenoble (France). Michel can be contacted at: riviere@us.ibm.com.

# Trademarks and special notices