



Remedies for memory-bound systems

IBM AIX 5L V5.3 system performance tuning series – Part 2



Hsian-Fen Tsao
IBM eServer Solutions Enablement
Mathew Accapadi
IBM AIX System Performance
January 2006

Table of contents

Abstract	1
Introduction	1
Performance tuning flowchart	1
Overview of AIX Virtual memory concepts	2
Remedies for memory-bound systems	4
Tuning the VMM.....	4
Tuning file memory/JFS2 file system cache limit	4
Tuning VMM page replacement	5
Example 1:	7
Different I/O configuration: Bypass file caching	7
Raw I/O.....	8
Direct I/O	8
Concurrent I/O	8
Release-behind mechanism for JFS and Enhanced JFS	9
Example 2:	9
Summary	10
Resources	10
About the authors	10
Trademarks and special notices	11

Abstract

This paper focuses on tuning memory-bound IBM® System p5™ and eServer™ pSeries® systems that are running under the IBM AIX 5L™ Version 5.3 operating system. It will first provide a brief overview of a common performance-tuning flowchart. The paper also gives a general introduction to virtual memory management (VMM) concepts. Finally, system performance tuning techniques for memory bottlenecks will be discussed and demonstrated by real-world case studies on an IBM POWER5™ processor-based system running the AIX 5L V5.3 operating system.

Introduction

A computer's main memory holds the program and data currently used by the central processing units (CPUs). When the system runs low on memory, it engages virtual memory, which involves reading and writing data to disks in the form of pages. Therefore, a shortage of memory will involve disk accesses for paging, resulting in performance degradation. When a secondary storage I/O occurs because of paging, the transaction response time is affected. Up to a point, memory bottlenecks can be resolved just by adding more RAM, which can be an easy solution, but is not the ideal first solution for cost reasons. Applying more cost-effective alternatives, such as tuning the operating system to remedy the memory bottlenecks, is usually a better first attempt at resolution before adding additional RAM.

Performance tuning flowchart

When investigating a performance bottleneck or tuning a system's performance, the first step is to identify where a bottleneck exists (for example: CPU, memory, disk, network, or application). The procedure for identifying a bottleneck usually starts with checking for CPU constraints, and then proceeds through the flowchart shown in Figure 1 shown on page 2.

The following is a general rule of thumb for bottleneck identification based on statistics gathered with the **vmstat** command and the performance tuning flowchart:

- The system is CPU-bound if the **sys** and **usr** categories are constantly greater than or equal to 90.
- If the system is not CPU-bound, the next area to examine is the memory usage. Often, an **iowait** time in excess of 20% (because of paging) can indicate a memory problem. If there is a high **pi** (page in) and **po** (page out) rate, it is likely that the system has memory constraints.
- If the system is not memory-bound, the next area to check for performance issues is the disk, and then the network.

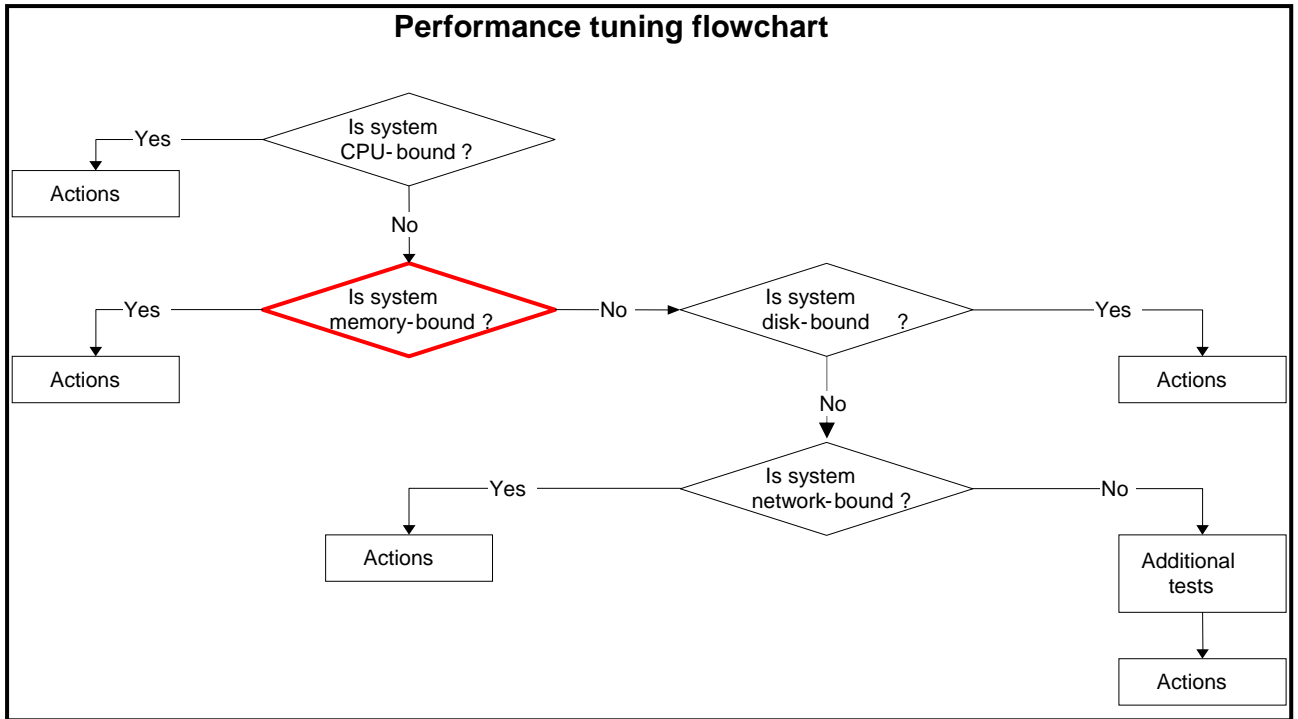


Figure 1: Performance tuning flowchart

Overview of AIX Virtual memory concepts

The AIX® operating system uses the virtual memory manager (VMM) to control memory and paging space. Virtual memory provides a methodology for making the real memory appear larger than its actual size. It is composed of the real memory plus physical disk space, where those portions of a file that are not currently in use are stored. The physical disk component of virtual memory is divided into three types of segments based on the location of the storage that is used to back the page in the following ways:

- Persistent segments are backed by a local file system on the pSeries system called the journaled file system (JFS).
- Working segments are backed by the paging space.
- Client segments are backed by a file system on a network file system (NFS), enhanced journaled file system (JFS2), or a Veritas® file system (VxFS).

Virtual memory segments are classified as containing either computational¹ or file² memory that is used to balance the types of pages stolen by the page-replacement algorithm. VMM is responsible for allocating real memory page frames and resolving references to pages that are not currently in real memory. VMM maintains a free list of available page frames. When the number of pages on the free list is low, VMM

¹ Computational memory consists of pages belonging to working storage or program text segments. It includes working storage memory of processes (stack, data, bss [basic subsystem]), the kernel text and data, kernel extensions, shared library data, and shared memory segments (that is, the DB2 Universal Database™ bufferpool).

² File memory consists of the remaining pages. These are usually pages from permanent data files in persistent storage or client storage. They can be normal data files or executables, but the AIX operating system gives preferential treatment to executables and, as such, treats executable file pages as computational pages.

uses a page-replacement algorithm through a page frame table (PFT) to determine which virtual memory pages that are currently in RAM will have their page frames reassigned to the free list.

The AIX 5L operating system provides support for both 32-bit and 64-bit user process memory models. User processes can use either 32-bit or 64-bit addresses. The 32-bit addresses provide some performance benefit because the instructions might be more compact. However, applications that need to access more than 2 gigabytes of data must use 64-bit addresses³.

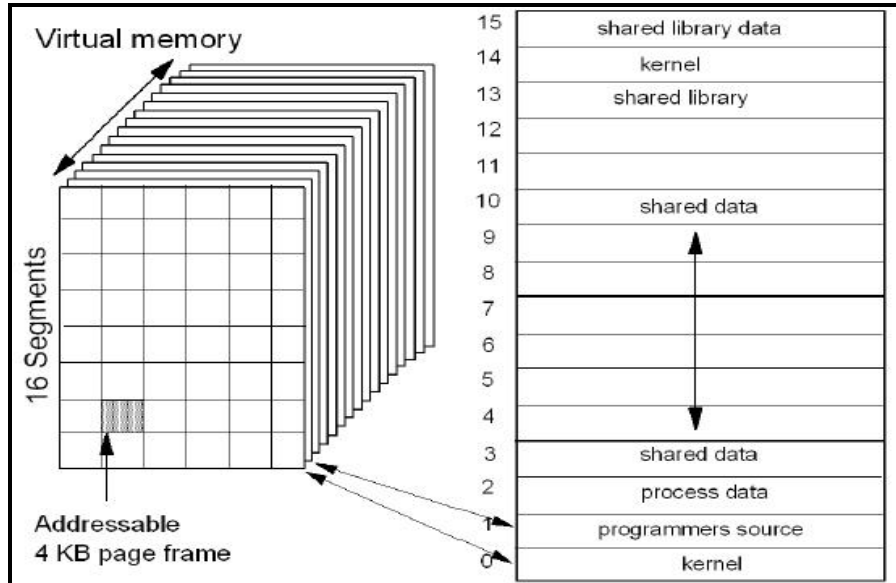


Figure 2: Virtual memory concepts

From the point-of-view of a 32-bit process, memory is further divided into 16 segments (Figure 2). Each process that is running uses these 16 segment registers. The segment registers are hardware registers located on the CPU. When a process is active, the registers contain the addresses of 16 segments that are addressable by the process. Each segment contains a specific type of information. For example, segment 0 is used for the kernel, segment 1 is used for the programmer's text, and segment 2 is used for process data. In the AIX 5L environment, virtual memory segments are partitioned into 4096-byte units called pages, although some systems also support larger page sizes (for example, 16 megabytes). Similarly, real memory is divided into 4096-byte page frames. Each page is either located in real physical memory (RAM) or is stored on disk until it is needed. When a page is required from a disk location, it is loaded into a page frame in real memory and a page fault is generated.

The function of the VMM from a performance point-of-view is to do the following:

- Minimize the processor usage and disk bandwidth that results from paging.
- Minimize the response degradation that results from paging because of a process

³ The 64-bit memory model can address up to 1 exabyte of memory. 256-megabyte memory segments are used as in the 32-bit memory model, but approximately 4.3 million segments are available instead of 16. Segments are dynamically allocated within specified ranges for various uses.

Remedies for memory-bound systems

A system is considered to be memory-bound when there are nonzero values for page-in and page-out rates, which are demonstrated by nonzero **pi** and **po** values in its **vmstat** reports. Memory-bound systems perform below expectations because disk reads and writes are slower than RAM reads and writes. The following memory bound remedies are intended to stop or reduce the virtual memory paging activity.

Tuning the VMM

There are several options for tuning the VMM for best performance.

Tuning file memory/JFS2 file system cache limit

The AIX 5L operating system uses real memory to accommodate page faults for computational and file pages. Computational memory includes the working pages. Noncomputational pages include persistent pages (pages of the JFS) and client pages (pages of file systems, such as JFS2, NFS client, and VxFS). Once file pages are in memory, they stay there until they are replaced by the page replacement algorithm; or the file pages are deleted from the file system; or the file system has been unmounted.

The amount of real memory that can be used for caching file pages is controlled by the following parameters listed in Table 1.

Parameter	Description
strict_maxperm	<ul style="list-style-type: none">• If 0 (default), then the limit for JFS pages is a soft limit.• If 1, then the limit for JFS2 pages is a hard limit and cannot exceed the value of maxperm%.• The value can be tuned via the command vmo -o strict_maxperm.
maxperm%	<ul style="list-style-type: none">• Percentage of memory used to cache JFS pages. The default is 80%.• Number of JFS pages can exceed this value if strict_maxperm is 0.• If the percent of memory occupied by the JFS pages exceeds maxperm%, when page replacement needs to occur, only file pages are replaced.• The value can be tuned via the command vmo -o maxperm.
strict_maxclient	<ul style="list-style-type: none">• If 1 (default), then the limit for JFS2, NFS client, and VxFS pages is a hard limit and cannot exceed the value of maxclient%.• If 0, then the number of these pages is a soft limit.• The value can be tuned via the command vmo -o strict_maxclient.
maxclient%	<ul style="list-style-type: none">• Percentage of memory used to cache JFS2, NFS client, and VxFS pages. The default is 80%.• A number of these pages can exceed this value if strict_maxclient is 0.• If the % of memory occupied by these pages exceeds maxclient%, when page replacement needs to occur, only these file pages are replaced.• Set to a value that is less than or equal to maxperm%, particularly in the case where the value of strict_maxperm is set to 1.• The value can be tuned via the command vmo -o maxclient.

Table 1: Tuning parameters to control file pages

Finding the appropriate value for the **maxperm** parameter ensures that the page replacement algorithm steals only file pages, because keeping computational pages in memory might be more

important for some workloads. When the **maxclient** threshold is reached, the least recently used (LRU) algorithm begins to steal client pages that have not been referenced recently. If not enough client pages can be stolen, then the LRU might replace other types of pages. Tuning the **maxclient** parameter depends on the AIX 5L release and the application's workload. Reducing the value for the **maxclient** parameter helps to prevent JFS2 file page access from causing the LRU to replace working storage pages.

Tuning VMM page replacement

The AIX 5L operating system keeps a list of free memory frames from which a frame is obtained to accommodate a page fault. There are also tunable parameters that determine the minimum number of free frames on this list, as well as a maximum number that determines when page replacement will stop running. If memory demand continues after reaching the minimum number of free frames on the list, then the processes might be killed.

Page replacement will run or stop when certain thresholds are reached, as shown in Table 2 below.

Threshold page replacement starts...	Threshold page replacement stops...
When the number of free page frames on the free list reaches minfree .	When the number of free page frames on the free list reaches maxfree .
If the number of JFS pages in memory is within minfree pages of maxperm and strict_maxperm is 1.	When the number of JFS pages is (maxperm-maxfree) .
If the number of JFS2 or other client pages in memory is within minfree pages of maxclient and strict_maxclient is 1.	When the number of client pages is (maxclient – maxfree) .
If Workload Manager is used and a WLM class has reached its memory limit.	

Table 2: Page replacement

When page replacement runs and LRU daemon is dispatched, the following occurs:

1. The page stealer scans the Page Frame Table⁴ (PFT) looking for unreferenced pages to steal with least recently used (LRU) algorithm.
2. If the page is referenced, then the least recently used daemon (**lrud**) will reset the reference bit to 0 and continue to the next page. Each time a page is examined to determine if it is a candidate for stealing or replacement, the scan value is incremented, which is expressed in the **sr** column of the **vmstat** output as pages scanned per second.
3. If the page is not referenced in the first pass, it is immediately stolen. The page can be stolen based on the criteria shown in Table 3.

⁴ A Page Frame Table (PFT) is an index of the contents for all real RAM pages that are held in pinned RAM. The PFT has flags to signal which pages have been referenced and which have been modified.

Criteria	Description
(A) numperm > maxperm	<ul style="list-style-type: none"> The page can be stolen if it is a file page. If the page is not a file page, it is left in memory.
(B) numperm <= maxperm and numperm >= minperm	<ul style="list-style-type: none"> Repaging rates are used to determine if the page can be stolen or replaced. <ul style="list-style-type: none"> A repage history buffer has 64 kilobytes of page entries that the VMM keeps. When a page is page-faulted in from disk, the page entry is stored in the repage history buffer. If the page is stolen from memory and is page-faulted in again from disk, the history buffer is checked to see if that entry is still in the repage history buffer. If in this case, a repage counter is incremented. There are two repage counters, one for computational pages, and one for file pages. If a file page was repaged (page-faulted in twice), then the file repage counter is incremented. If a computational page was repaged, then the computational repage counter is incremented. If the file repage counter is higher than the computational repage counter, then computational pages are stolen. If the computational repage counter is higher than the file repage counter, then file pages are stolen. If the lru_file_repage parameter is set to 0, then the repage counters are not used to determine what type of pages to steal.
(C) numperm < minperm	<ul style="list-style-type: none"> The lrud daemon does not care what type of page it is. This page can be stolen if it is unreferenced.
(D) lru_file_repage=0	<ul style="list-style-type: none"> As long as numperm is above minperm, only file pages will be stolen. (Criterion B is ignored.)

Table 3: Criteria for stealing pages

The **lru** daemon uses the concept of LRU buckets to scan memory. Memory is divided into buckets of **lrubucket** pages. The default is 131072 pages or 512 megabytes. The daemon scans a bucket looking for pages to steal (and resetting the reference bits at the same time). After the end of the bucket is reached, if not enough pages can be stolen, it starts over on that same bucket rather than continuing with the rest of the memory pages. This ensures that not too much scanning is done. If in the second pass of the bucket, not enough pages can be stolen, then the **lru** daemon will continue to the next bucket.

By default, file pages can be cached in real memory for file systems. The caching can be disabled using direct I/O or concurrent I/O mount options; also, the Release-Behind mount options can be used to quickly discard file pages from memory after they have been copied to the application's I/O buffers if the read-ahead and write-behind benefits of cached file systems are needed. These tuning techniques will be discussed in the next section.

Example 1:

Problem scenario: An enterprise runs the application on a POWER5 processor-based system with the AIX 5L V5.3 operating system installed. The database is on JFS2, and there is paging out to paging space (Figure 3).

```
System Configuration: lcpu=24 mem=92160MB
```

kthr		memory			page				faults			cpu				
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
10	82	7922525	3045	0	39	3472	40727	55082	0	2951	217026	54604	36	34	0	30
21	70	7922873	3057	0	9	2076	42936	53779	0	2858	219840	53694	38	34	0	28
47	43	7921948	2982	0	23	756	40141	80523	0	2636	224916	49550	39	34	0	27
46	44	7921431	2978	0	12	252	40710	114595	0	2622	224561	43536	38	33	0	29
66	24	7921400	3176	0	5	48	35612	62075	0	2127	190123	41881	37	29	0	35

Figure 3: Paging space is occurring

Diagnosis: The **pi** and **po** columns in **vmstat** output are constantly nonzero because database file pages are causing database buffer cache pages to get paged out.

Tuning description: Disable the use of repage counters and reduce the **minperm%** parameter (via the **vm tune** or **vmo** command) to ensure that the **numclient** parameter is higher than the **minperm** parameter.

minperm%=5

lru_file_repage=0

Tuning results: The paging condition was eliminated.

Different I/O configuration: Bypass file caching

The AIX 5L operating system uses file caching as the default method of file access. The contents of the buffer are cached in RAM through the VMM's use of real memory as a file buffer cache. This method is based on the fact that the access to memory is much faster than to the disk. However, file caching consumes more CPU resources and a significant portion of system memory because of its data duplication. The file buffer cache can improve I/O performance for workloads with a high hit ratio on the file system. Some databases tend to work better when using the file cache. Here are two examples:

- The nature of 32-bit database applications is such that the buffer cache size is limited.
- For database applications that do a lot of table scans for tables that are much larger than the database buffer cache, file system readahead can help.

But certain classes of applications derive no benefit from the file buffer cache. Using the database workload as an example, databases normally manage data caching at the application level. Because of this, they do not need the file system to implement this service for them. The use of a file buffer cache results in undesirable overheads in such cases because the file system cache hit ratio is very low. In these situations, memory can often be better utilized by the database application.

Raw I/O

Database applications traditionally use raw logical volumes instead of the file system for performance reasons. Writes to a raw device bypass the caching, logging, and inode locks that are associated with the file system; data gets transferred directly from the application buffer cache to the disk. If an application is update-intensive with small I/O requests, then raw device setup for database data and logging can help performance and reduce the usage of memory resources. The asynchronous I/O (AIO) Fastpath is used only on raw logical volumes. But raw I/O presents a challenge for data storage administration. In contrast, the file system storage is easier to maintain and manage but there is a possible tradeoff in lower performance, especially for common database workloads such as online transaction processing (OLTP). You can use the AIX 5L command **mklv** with the option **-t raw** to set up the raw device.

Direct I/O

In 2003, the AIX 5L V5.2 operating system introduced direct I/O (DIO) for the enhanced JFS system (JFS2). DIO is similar to raw I/O except DIO is supported under a file system, whereas raw I/O is supported by writing to the **/dev/rdisk** device. They both bypass the file system buffer cache, which reduces CPU overhead and makes more memory available to others (that is, to the database instance). DIO has similar performance benefit as raw I/O but is easier to maintain for the purposes of system administration. For applications that need to bypass the buffering of memory within the file system cache, direct I/O is provided as an option in JFS2 and JFS. For instance, some technical workloads never reuse data because of the sequential nature of their data access. This lack of data reuse results in a poor buffer cache hit rate, which means that these workloads are good candidates for DIO. By using the **mount** command with the **-o dio** option specified, all files in the file system use DIO by default.

For direct I/O to work, files must be accessed with the correct offset alignment and transfer size according to the file system used. Failure to meet these requirements will cause direct I/O to be demoted, and the data will end up going through VMM, which can cause a performance penalty. Although DIO eliminates the file system buffer cache overhead, the write-exclusive inode lock can still be a performance bottleneck. This inode lock can result in threads being blocked during context switches if the application is update-intensive.

Concurrent I/O

JFS2 supports concurrent file access to files. In addition to bypassing the file cache, it also bypasses the inode lock that allows multiple threads to perform reads and writes simultaneously on a shared file. Concurrent I/O (CIO) is designed for relational database applications, most of which will operate under concurrent I/O without any modification. Applications that do not enforce serialization for access to shared files should not use CIO. Applications that issue a large amount of reads usually will not benefit from CIO either.

All files in a file system use CIO by default when invoking the **mount** command with the **-o cio** option specified. Files that directly use CIO utilize the DIO path, and these files are subject to the same alignment and transfer size as the DIO. Implicitly, the **-o dio** flag is overridden for these files.

Note: DIO and CIO can be restricted to a subset of files in a file system by placing the files that require these I/O techniques in a separate subdirectory and using **namefs** to mount this subdirectory over the file system. For example, if a file system **somefs** contains some files that prefer to use DIO and CIO, as well as others that do not, you can create a subdirectory, **subsomefs**, in which you place all the files that require DIO or CIO. Mount **somefs** without specifying **-o dio**, and then mount **subsomefs** as a **namefs** file system with the **-o dio** option using the following command:

```
mount -v namefs -o dio /somefs/subsomefs /someotherfs/subsomefs
```

Release-behind mechanism for JFS and Enhanced JFS

Release-behind-read and release-behind-write allow the file system to release the file pages from file system buffer cache as soon as an application has read or written the file pages. This feature helps the performance when an application performs a great deal of sequential reads or writes. Most often, these file pages will not be reassessed after they are accessed.

Without this option, the memory will still be occupied with no benefit of reuse, which causes paging eventually after a long run. When writing a large file without using release-behind, writes will go very fast as long as pages are available on the free list. When the number of pages drops to **minfree**, VMM uses its LRU algorithm to find candidate pages for eviction.

This feature can be configured on a file system basis. When using the **mount** command, enable release-behind by specifying one of the three flags below:

- The release-behind sequential read flag (rbr)
- The release-behind sequential write flag (rbw)
- The release-behind sequential read and write flag (rbrw)

A trade-off of using the release-behind mechanism is that the application can experience an increase in CPU utilization for the same read or write throughput rate (as compared to not using release-behind). This is because of the work required to free the pages, which is normally handled at a later time by the LRU daemon. Also, note that all file page accesses result in disk I/O because file data is not cached by VMM. However, applications (especially long-running applications) with the release-behind mechanism applied are still supposed to perform more optimally and with more stability.

Example 2:

Problem scenario: A solution provider runs an application on a POWER5 processor-based system with the AIX 5L V5.3 operating system installed. The application and database are on JFS2. There is paging out to paging space after several hours of long runs.

Diagnosis: The application performs a great deal of sequential reads and writes. Afterward, these pages will not be reaccessed for a while. After several long runs, database file pages are causing database buffer cache pages to get paged out.

Tuning description: Mount the database with release-behind sequential read (-rbr) on a file system in which the access pattern is primarily a read pattern.

Tuning results: The paging condition was eliminated.

Summary

Tuning a memory-bound system requires analysis of performance data and knowledge of the application I/O patterns; if the application I/O patterns are not known, then additional data, such as trace data, can be gathered to determine this. Note that improper system tuning can sometimes result in unexpected system behavior, such as performance degradation. Therefore, changes need to be applied only when a bottleneck has been identified. The AIX 5L operating system provides a lot of very useful performance tools and features that can be used to monitor, identify, and remedy the memory bottleneck. The AIX 5L tuning techniques discussed in this paper have been proven to be effective on memory-bound systems in various IBM solution provider and enterprise environments.

Resources

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

- IBM eServer p5 Information Center
<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp>
- IBM Publications Center
www.elink.ibm.com/public/applications/publications/cgi-bin/pbi.cgi?CTY=US
- IBM Redbooks
www.redbooks.ibm.com/
- Improving database performance with AIX® concurrent I/O
[ibm.com/servers/aix/whitepapers/db_perf_aix.pdf](http://www.ibm.com/servers/aix/whitepapers/db_perf_aix.pdf)
<http://www3.software.ibm.com/ibmdl/pub/software/dw/dm/db2/dm-0408lee/CIO-article.pdf>
- AIX 5L V5.3 performance management guide and AIX 5L V5.3 performance tools guide and reference
<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp>

About the authors

Hsian-Fen Tsao is an IBM technical consultant for eServer pSeries and AIX software vendors. Before joining the IBM Solution Enablement Group in Austin, Texas, she worked in the pSeries AIX Performance Group for 10 years. Her experiences include database (online transaction processing and decision support systems), Web server, TCP/IP, Java™ and IBM WebSphere® technologies, and the POWER5 Virtualization Engine™. You can contact her at htsao@us.ibm.com.

Mathew Accapadi is an IBM senior technical staff member (STSM) in AIX performance in the System & Technology Group. He has 18 years of AIX experience and works on improving the performance of the AIX operating system and applications that run in this environment. You can reach him at accapadi@us.ibm.com.

Trademarks and special notices

© IBM Corporation 1994-2006. All rights reserved.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	eServer	DB2	DB2 Universal Database
ibm.com	Redbooks	WebSphere	Virtualization Engine
the IBM logo	pSeries	AIX	AIX 5L

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.