

Paging Space Policies and Practice

February 27, 2001

Beth Morton

Mathew Accapadi

Kim Tran

Maritza Borunda



Paging Space Policies and Practice



Paging Space Policies and Practice

Contents

Paging Space Policies 1

Performance Overview of the Virtual Memory Manager (VMM)	1
Real-Memory Management	1
Understanding Paging Space Allocation Policies	5
Late Allocation Algorithm	5
Early Allocation Algorithm	6
Deferred Allocation Algorithm	6

Paging Space Practice 7

Placement and Sizing of Paging Space	7
Size Considerations for Systems with Large Amounts of RAM.	8
Size Considerations When Using a Dedicated Dump Device	8
Monitoring Your Paging Space	8
Tools Available to Help You Monitor Your Paging Space.	8

Using the vmstat Command	8
Using the ps Command	11
Using the svmon Command	11
System Configuration Examples	12
Configuring a Database Server That Uses Raw LVM Files	12
Configuring a Database Server That Uses Filesystem Files	13
Configuring a System That Has Real-Time Requirements.	13
Configuring a System That Has a Small Amount of Memory	14

Appendix A. Commands Available to Manage and Monitor Your Paging Space 15

Appendix B. Special Notices 17

Paging Space Policies

This paper is intended for system administrators who are interested in evaluating the use of paging space. The paper is presented in two parts. The first part provides background information on paging space and the Virtual Memory Manager (VMM). If you already have an understanding of these principles, you can skip ahead to part two. The second part of this paper provides rules-of-thumb for approximating paging space. It also describes tools that are available to evaluate and tune paging space needs.

A paging space is fixed disk storage that is used for temporary storage of pages when there isn't enough space in real memory. A paging space, also called a *swap space*, is a logical volume with the attribute type equal to paging. This type of logical volume is referred to as a paging-space logical volume, or simply *paging space*. When the amount of free real memory in the system is low, programs or data that have not been used recently are moved from real memory to paging space to release real memory for other activities.

The following topics are discussed in this paper:

- **Paging Space and the Virtual Memory Manager Principles**
 - “Performance Overview of the Virtual Memory Manager (VMM)”
 - “Real-Memory Management”
 - “Understanding Paging Space Allocation Policies” on page 5
- **Paging Space Practice**
 - “Placement and Sizing of Paging Space” on page 7
 - “Monitoring Your Paging Space” on page 8
 - “Tools Available to Help You Monitor Your Paging Space” on page 8

For detailed information about the AIX operating system, refer to the following Web address: <http://www.ibm.com/servers/aix/library/>. AIX library information is listed under *Technical Publications*.

Performance Overview of the Virtual Memory Manager (VMM)

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

Real-Memory Management

Virtual-memory segments are partitioned into fixed-size units called *pages*. The page size is 4096 bytes. Each page in a segment can be in real memory (RAM), or stored on disk until it is needed. Similarly, real memory is divided into 4096-byte page frames. The role of the VMM is to manage the allocation of real-memory page frames and to resolve references by the program to virtual-memory pages that are not currently in real memory or do not yet exist (for example, when a process makes the first reference to a page of its data segment).

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

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

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

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

Free List

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

Persistent versus Working Segments

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

Persistent segment types are further classified.

- *Client segments* are used to map remote files (for example, files that are being accessed through NFS), including remote executable programs. Pages from client segments are saved and restored over the network to their permanent file location, not on the local-disk paging space.
- *Journalled* and *deferred segments* are persistent segments that must be atomically updated. If a page from a journalled or deferred segment is selected to be removed from real memory (paged out), it must be written to disk paging space unless it is in a state that allows it to be committed (written to its permanent file location).

Working segments are transitory, exist only during their use by a process, and have no permanent disk-storage location. Process stack and data regions are mapped to working segments, as are the kernel text segment, the kernel-extension text segments, as well as the shared-library text and data segments. Pages of working

segments must also have disk-storage locations to occupy when they cannot be kept in real memory. The paging space is used for this purpose.

Computational versus File Memory

Computational memory, also known as computational pages, consists of the pages that belong to working-storage segments or program-text segments. *File memory*, or file pages, consists of the remaining pages. These are usually pages from permanent data files in persistent storage.

Page Replacement

When the number of available real memory frames on the free list becomes low, the page stealer is invoked. The page stealer moves through the Page Frame Table (PFT), looking for pages to steal. The PFT includes flags to signal which pages have been referenced and which have been modified. If the page stealer encounters a page that has been referenced, it does not steal that page, but instead, resets the reference flag for that page. The next time the clock hand (page stealer) passes that page and the reference bit is still off, that page is stolen. A page that was not referenced in the first pass is immediately stolen.

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

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

When a process exits, its working storage is released immediately and its associated memory frames are put back on the free list. However, any files that the process may have opened can stay in memory until they are stolen by VMM or until the corresponding file is deleted.

Page replacement is done directly within the scope of the thread if running on a uniprocessor. On a multiprocessor system, page replacement is done through the **lrud** kernel process, which is dispatched to a CPU when the **minfree** threshold has been reached. Beginning with AIX 4.3.3, the **lrud** kernel process is multithreaded with one thread per memory pool. Real memory is split into evenly sized memory pools based on the number of CPUs and the amount of RAM. The number of memory pools will be as follows:

MAX (Number of CPUs/8, RAM in GB/16), but not more than the number of CPUs and not less than 1

AIX 4.3.3 (and later) uses the **vmtune -n number_of_memory_pools** command to change the number of memory pools that will be configured at system boot. The values for **minfree** and **maxfree** in the **vmtune** command output will be the sum of the **minfree** and **maxfree** for each memory pool. For more information on **minfree** and **maxfree**, see “VMM Thresholds” on page 4.

Note: The **vmtune -n** is only available on multiprocessor systems.

Repaging

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

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

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

VMM Thresholds

Several numeric thresholds define the objectives of the VMM. When one of these thresholds is exceeded, the VMM takes appropriate action to bring the state of memory back within bounds. This section discusses the thresholds that the system administrator can alter through the `/usr/samples/kernel/vmtune` command.

The number of page frames on the free list is controlled by the following parameters:

minfree

Minimum acceptable number of real-memory page frames in the free list. When the size of the free list falls below this number, the VMM begins stealing pages. It continues stealing pages until the size of the free list reaches **maxfree**.

maxfree

Maximum size to which the free list will grow by VMM page-stealing. The size of the free list may exceed this number as a result of processes terminating and freeing their working-segment pages, or deleting files that have pages in memory.

The VMM attempts to keep the size of the free list greater than or equal to **minfree**. When page faults or system demands cause the free list size to fall below **minfree**, the page-replacement algorithm runs. The size of the free list must be kept above the default value of **minfree** for several reasons. For example, the AIX sequential-prefetch algorithm requires several frames at a time for each process that is doing sequential reads. Also, the VMM must avoid deadlocks within AIX itself, which could occur if there were not enough space to read in a page that was required to free a page frame.

The following thresholds are expressed as percentages. These thresholds represent the fraction of the total real memory of the machine that is occupied by file pages (pages of noncomputational segments).

minperm

If the percentage of real memory occupied by file pages falls below this level, the page-replacement algorithm steals both file and computational pages, regardless of repage rates.

maxperm

If the percentage of real memory occupied by file pages rises above this level, the page-replacement algorithm steals only file pages.

numperm

Gives the number of file pages in memory.

When the percentage of real memory occupied by file pages is between **minperm** and **maxperm**, the VMM normally steals only file pages, but if the repaging rate for file pages is higher than the repaging rate for computational pages, computational pages are stolen as well.

The main intent of the page-replacement algorithm is to ensure that computational pages are given fair treatment. For example, the sequential reading of a long data file into memory should not cause the loss of program text pages that are likely to be used again soon. The page-replacement algorithm's use of the thresholds and repaging rates ensures that both types of pages get treated fairly, with a slight bias in favor of computational pages.

For more information on **minperm** and **maxperm**, see "Choosing **minperm** and **maxperm** Settings" in *AIX 5L Version 5.1 Performance Management Guide*. You can find this guide at <http://www.ibm.com/servers/aix/library/>.

Note: It takes experience and know-how to set **vmtune** parameters properly. Therefore, it is highly recommended that you set **vmtune** values only after consulting the AIX Performance Team. Contact your AIX Support Center and ask for a performance analysis.

Understanding Paging Space Allocation Policies

AIX supports three allocation methods for working storage, also referred to as *paging-space slots*, as follows:

- Late allocation
- Early allocation
- Deferred allocation

Note: Paging-space slots are only released by process (not by thread) termination or by the **disclaim** subroutine. The slots are not released by the **free** system call.

Late Allocation Algorithm

Prior to AIX 4.3.2, with the Late Page Space Allocation (LPSA) policy, a paging slot is allocated to a page of virtual memory only when that page is first read from or written into; that is, the first time that the page's content is of interest to the executing program.

Many programs exploit late allocation by allocating virtual-memory address ranges for maximum-sized structures and then only using as much of the structure as the situation requires. The pages of the virtual-memory address range that are never accessed never require real-memory frames or paging-space slots.

This technique does involve some degree of risk. If all of the programs running in a machine happened to encounter maximum-sized situations simultaneously, paging space might be exhausted and some programs might not be able to continue to completion.

Early Allocation Algorithm

The second paging-space slot-allocation method is intended for installations where this situation is likely, or where the cost of failure to complete is intolerably high. The early allocation algorithm causes the appropriate number of paging-space slots to be allocated at the time the virtual-memory address range is allocated, for example, with the **malloc** subroutine. If there are not enough paging-space slots to support the **malloc** subroutine, an error code is set. The early-allocation algorithm is invoked as follows:

```
export PSALLOC=early
```

This example causes all future programs executed in the environment to use early allocation. The currently executing shell is not affected.

Early allocation is of interest to the performance analyst mainly because of its paging-space size implications. If early allocation is turned on for those programs, paging-space requirements can increase many times more than the recommended size.

For recommended paging space size, see “Placement and Sizing of Paging Space” on page 7.

Deferred Allocation Algorithm

Deferred Page Space Allocation (DPSA) policy delays allocation of paging space until it is necessary to page-out the page, which results in no wasted paging space allocation. This method can save large amounts of disk space. DPSA is the default paging space method beginning with AIX 4.3.2.

On some systems, paging space might not ever be needed even if all the pages accessed have been touched. This situation is most common on systems with a very large amount of RAM. However, this can result in overcommitment of paging space in cases where more virtual memory than available RAM is accessed.

To disable DPSA and preserve the Late Page Space Allocation policy, run the following command:

```
/usr/samples/kernel/vmtune -d 0
```

To activate DPSA, run the following command:

```
/usr/samples/kernel/vmtune -d 1
```

Paging Space Practice

The following sections describe paging space practice:

- “Placement and Sizing of Paging Space”
- “Monitoring Your Paging Space” on page 8
- “Tools Available to Help You Monitor Your Paging Space” on page 8

Placement and Sizing of Paging Space

The amount of paging space required depends on the type of activities performed on the system and the amount of available memory. If paging space runs low, processes may be lost, and if paging space runs out, the system can panic. When a paging-space low condition is detected, additional paging space should be defined.

The system monitors the number of free paging-space blocks and detects when a paging-space shortage exists. When the number of free paging-space blocks falls below a threshold known as the *paging-space warning level*, the system informs all processes (except **kprocs**) of this condition by sending the SIGDANGER signal. When paging space is low, you might receive any of the following messages:

```
INIT: Paging space is low!  
ksh: cannot fork no swap space  
Not enough memory  
Fork function failed  
fork () system call failed  
Unable to fork, too many processes  
Fork failure - not enough memory available  
Fork function not allowed. Not enough memory available.
```

The default paging space size is determined when the system is configured during installation, according to the following standards:

- Set paging space to 2 times the amount of RAM
- Paging space can use no more than 20% of total disk space in the root volume group
- Paging space can be no larger than 2 GB

Ideally, there should be several paging spaces of roughly equal size, each on a different physical disk drive. If you decide to create additional paging spaces, create them on physical volumes that are more lightly loaded than the physical volume in **rootvg**. While the system is booting, only the primary paging space (**hd6**) is active. Consequently, all paging-space blocks allocated during system installation are on the primary paging space. The primary paging space should therefore be somewhat larger than the secondary paging spaces. The secondary paging spaces should all be of the same size to ensure that the algorithm performed can work effectively.

The **lsps -s** command gives a snapshot of the current utilization level of all the paging spaces on a system.

Size Considerations for Systems with Large Amounts of RAM

One of the purposes for having a large amount of RAM is to decrease the amount of paging. If your system has a large amount of RAM (2 GB or more), the rules for doubling the amount of RAM for paging space might not apply for your system. To approximate the amount of paging space you'll need, run the tools described below at peak performance time to review how much paging space is being used. Then set the paging space accordingly.

For more information on configuring paging space, see "Adding and Activating a Paging Space" in *AIX 5L Version 5.1 System Management Guide: Operating Systems and Devices*.

Size Considerations When Using a Dedicated Dump Device

If you decide on a paging space policy that reserves a minimum amount of paging space, you should consider using a dedicated dump device. If your paging space is small and the system needs to perform a system dump, there might not be enough space available to contain the dump. Configuring a dedicated dump device helps ensure that you will have enough space to contain the system dump.

Monitoring Your Paging Space

One way to determine the appropriate amount of RAM for a system is to look at the largest value for `avm` (active as reported by the `vmstat` command). Multiply that value by 4K to get the number of bytes. Compare that result to the number of bytes of RAM on the system. Ideally, `avm` should be smaller than total RAM—if not, some amount of virtual memory paging will occur. How much will depend on the difference between the two values.

Virtual memory gives us the capability of addressing more memory than we have (some of the memory is in RAM and the rest is in paging space). But if there is far more virtual memory than real memory, this could cause excessive paging, which then results in delays. If `avm` is lower than RAM, then paging space paging could be caused by RAM being filled up with file pages. In this case, tuning the `minperm` and `maxperm` values could reduce the amount of paging.

Tools Available to Help You Monitor Your Paging Space

The following commands are available to assist you in tuning your paging space:

- `vmstat`
- `ps`
- `svmon`

Using the `vmstat` Command

The `vmstat` command summarizes the total *active* virtual memory used by all of the processes in the system, as well as the number of real-memory page frames on the free list. Active virtual memory is defined as the number of virtual-memory working segment pages that have actually been touched. This number can be larger than the number of real page frames in the machine, because some of the active virtual-memory pages may have been written out to paging space.

When determining if a system might be short on memory or if some memory-tuning needs to be done, run the `vmstat` command over a set interval and examine the `pi` and `po` columns on the resulting report. These columns indicate the number of paging space page-ins per second and the number of paging space

page-outs per second. If the values are constantly non-zero, there might be a memory bottleneck. Having occasional non-zero values is not be a concern, because paging is the main principle of virtual memory.

In the following report, notice the high I/O wait in the output and also the number of threads on the blocked queue. Most likely, the I/O wait is due to the paging in and paging out from paging space.

```
# vmstat 2 10
kthr      memory          page          faults          cpu
-----
r  b   avm   fre  re  pi  po  fr  sr  cy  in  sy  cs  us  sy  id  wa
1  3 113726  124  0  14   6 151 600  0 521 5533 816 23 13  7 57
0  3 113643  346  0   2  14 208 690  0 585 2201 866 16  9  2 73
0  3 113659  135  0   2   2 108 323  0 516 1563 797 25  7  2 66
0  2 113661  122  0   3   2 120 375  0 527 1622 871 13  7  2 79
0  3 113662  128  0  10   3 134 432  0 644 1434 948 22  7  4 67
1  5 113858  238  0  35   1 146 422  0 599 5103 903 40 16  0 44
0  3 113969  127  0   5  10 153 529  0 565 2006 823 19  8  3 70
0  3 113983  125  0  33   5 153 424  0 559 2165 921 25  8  4 63
0  3 113682  121  0  20   9 154 470  0 608 1569 1007 15  8  0 77
0  4 113701  124  0   3  29 228 635  0 674 1730 1086 18  9  0 73
```

To see if the system has performance problems with its VMM, examine the columns under memory and page:

- **memory**

Provides information about the real and virtual memory.

- **avm**

Average number of 4K pages that are allocated to paging space. The avm (Active Virtual Memory) value can be used to calculate the amount of paging space assigned to executing processes.

Note: Starting with AIX 4.3.2, there is a slight change in reporting this value.

The number in the avm field divided by 256 yields the approximate number of megabytes (MB) allocated to paging space systemwide. Prior to AIX 4.3.2, the same information is reflected in the Percent Used column of the **lsps -s** command output or with the **svmon -G** command under the pg space inuse field.

- **fre**

Average number of free memory pages. A page is a 4 KB area of real memory.

When an application terminates, all of its working pages are immediately returned to the free list. Its persistent pages (files), however, remain in RAM and are not added back to the free list until they are stolen by the VMM for other programs. Persistent pages are also freed if the corresponding file is deleted.

For this reason, the fre value might not indicate all the real memory that can be readily available for use by processes. If a page frame is needed, persistent pages related to terminated applications are among the first to be handed over to another program.

If the fre value is substantially above the **maxfree** value, it is unlikely that the system is *thrashing* (the system is continuously paging in and out). However, if the system is experiencing thrashing, the fre value will be small.

- **page**
Provides information about page faults and paging activity, averaged over the interval and given in units per second.
 - **re**
The re column shows reclaiming, which is no longer supported in AIX Version 4. The algorithm to keep track of reclaims costs performance, and the value delivered does not provide very useful information about system performance.
 - **pi**
Number (rate) of pages paged in from paging space. Paging space is the part of virtual memory that resides on disk. It is used as an overflow when memory is overcommitted. Paging space consists of logical volumes dedicated to the storage of working set pages that have been stolen from real memory. When a stolen page is referenced by the process, a page fault occurs, and the page must be read into memory from paging space.
Due to the variety of configurations of hardware, software, and applications, there is no absolute number to look out for. But five page-ins per second per paging space should be the upper limit. This guideline should not be rigidly adhered to, but used only as a reference. This column is important as a key indicator of paging-space activity. If a page-in occurs, there must have been a previous page-out for that page. It is also likely, in a memory-constrained environment, that each page-in will force a different page to be stolen and, therefore, paged out. But systems could also operate efficiently when they have close to 10 pi per second for 1 minute and then work without any page-ins.
 - **po**
Number (rate) of pages paged out to paging space. Whenever a page of working storage is stolen, it is written to paging space, if it does not yet reside in paging space or if it was modified. If not referenced again, it will remain on the paging device until the process terminates or disclaims the space. Subsequent references to addresses contained within the faulted-out pages result in page faults, and the pages are paged in individually by the system. When a process terminates normally, any paging space allocated to that process is freed. If the system is reading in a significant number of persistent pages (files), you might see an increase in po without corresponding increases in pi. This does not necessarily indicate thrashing, but might warrant investigation into data-access patterns of the applications.
 - **fr**
Number of pages that were freed per second by the page-replacement algorithm during the interval. As the VMM page-replacement routine scans the Page Frame Table (PFT), it uses criteria to select which pages are to be stolen to replenish the free list of available memory frames. The criteria include both working (computational) and file (persistent) pages. A page being freed does not mean that any I/O has taken place. For example, if a persistent storage (file) page has not been modified, it will not be written back to the disk. If I/O is not necessary, minimal system resources are required to free a page.
 - **sr**
Number of pages that were examined per second by the page-replacement algorithm during the interval. The VMM page-replacement code scans the PFT and steals pages until the number of frames on the free list is at least the **maxfree** value. The page-replacement code might have to scan many entries in the PFT before it can steal enough to satisfy the free list requirements. With

stable, unfragmented memory, the scan rate and free rate might be nearly equal. On systems with multiple processes using many different pages, the pages are more volatile and disjointed. In this scenario, the scan rate might greatly exceed the free rate.

Memory is overcommitted when the ratio of fr to sr (fr:sr) is high.

An fr:sr ratio of 1:4 means that for every page freed, four pages had to be examined. It is difficult to determine a memory constraint based on this ratio alone. A high ratio is workload- and application-dependent.

– **cy**

Number of cycles per second of the clock algorithm, a technique used by VMM to select pages to be replaced. This technique takes advantage of a referenced bit for each page as an indication of what pages have been recently used (referenced). When the page-stealer routine is called, it cycles through the PFT, examining each page's referenced bit.

The cy column shows how many times per second the page-replacement code has scanned the PFT. Because the free list can be replenished without a complete scan of the PFT and because all of the **vmstat** fields are reported as integers, this field is usually zero. If not, it indicates a complete scan of the PFT, and the stealer has to scan the PFT again, because fre is still under the **maxfree** value.

Using the ps Command

You can get a general idea of which processes are using the most memory from the **ps** command. Although less detailed and only a snapshot in time, **ps** still provides some useful information.

Using the svmon Command

The **svmon** command provides a more in-depth analysis of memory usage, by capturing a snapshot of the current state of memory. The display information can be analyzed using four different reports:

-G (Global)

Displays statistics describing the real memory in use and paging space in use for the whole system.

-P (process)

Displays memory usage statistics for active processes.

-S (segment)

Displays memory usage statistics for specified segments or for the top ten segments.

-D (detailed segment)

Displays detailed information on specified segments, including specific pages in use and their status.

The **svmon** command can be run in intervals with the **-i** flag.

svmon Examples

- To print out global statistics:
`svmon -G`
- To print out the memory usage statistics for all processes (limit statistics to non-system segments):
`svmon -Pn`

- To print out the memory usage statistics for processes 6746 and 10078 (include statistics on all segments):
svmon -Pa 6746 10078
- To sort system segments by the number of pages in real memory, and print out the top 10 segments of the resulting list:
svmon -Ssu 10
- To print detailed memory usage statistics on segment 340d:
svmon -D 340d
- To print out the global statistics, repeated 10 times at 30 second intervals:
svmon -G -i 30 10

Beginning with AIX 4.3.3, the following is the output from **svmon -G**:

	size	inuse	free	pin	virtual
memory	32744	32412	332	3630	52227
pg space	80896	34549			
	work	pers	clnt		
pin	3630	0	0		
in use	22970	9414	28		

The size and inuse columns of the pg space output represent paging space usage. The size is measured as the number of 4K pages.

System Configuration Examples

The following examples illustrate common system configurations, as well as detailing workloads. Solutions are provided to assist you in evaluating your own system configurations to improve system performance.

Configuring a Database Server That Uses Raw LVM Files

Scenario

- Your system is used purely as a database server; that is, there are no users logged in and no other applications running on it.
- The database files are raw LVM files and do not contain filesystem files.
- There are a fixed number of database processes and threads running.

Because the number of processes and threads are fixed, and because there will be no file caching (database files are raw LVM files, so the caching is done by the database itself in its database buffer pool), it is possible to configure the system so that it will never have to page-in or page-out. In this case, the amount of paging space can be kept at a minimum. Configure enough paging space to allow the system to run without killing processes but still provide the system administrator enough time to take action against the programs that have memory leaks.

Solution

1. Determine the memory requirements.
Boot the system and start the database, then calculate the current amount of memory being used. To do this, either run the **vmstat** command and look at the **avm** column, or run the **svmon** command and look at the **inuse** value for memory. Multiply the result by 4 KB to get the total amount of memory in bytes.
2. Determine the size of the database buffer cache.
Multiply the number of database buffer blocks by the size of the database buffer blocks. These are parameters to the database and can be found in the

database initialization file. This is done to determine if the database buffers are actually used. If the database buffers are not used, they do not use memory.

3. Determine how many additional database processes will be running on this system.

This can be based on the number of users or on how many processes are configured by a parameter in the database initialization file. Start one user or database process and measure its memory usage using either the `ps -v processid` command or the `svmon -P processid` command. Then multiply the memory usage of a process by the total number of database processes you expect to have.

4. Sum the values from steps 1–3 to obtain the total memory requirement.

Compare this value to the amount of real memory on the system. If this is less than the amount of real memory, then your paging space can be small. If it's more than the amount of real memory, then ideally the number of database blocks should be reduced so that it is less than the amount of real memory, or you should have at least as much paging space as the difference between this calculated value and the total amount of real memory.

Configuring a Database Server That Uses Filesystem Files

Scenario

This scenario is similar to the previous example, except that the database files are filesystem files and not raw LVM files. In this case, double-buffering occurs; that is, the database caches contents of the database files in its own database buffer cache, and the VMM caches the files in the VMM file cache.

Solution

If you want to use filesystem files, tune the system with the `vmtune` command. The values of `maxperm` and `minperm` should be reduced until they will be less than the `numperm` value. If you keep `maxperm` and `minperm` very small (approximately 1%), you can use the same method as in the previous example to determine your paging space requirement.

If you do not do this tuning, keep in mind that the database buffer cache could be paged out because the VMM file cache is also caching the database files. This would require the use of real memory, leaving less memory for the database buffer cache. In this case, you should have at least as much paging space as you have allocated for the database buffer cache plus the memory usage of all the database processes.

Configuring a System That Has Real-Time Requirements

Scenario

The system has some real-time requirements. For example, it could be running High Availability Cluster Multi-Processing (HACMP), which requires real-time response on the behalf of the cluster-manager daemon (`clstrmgr`).

Note: The Cluster Manager daemon (`/usr/sbin/cluster/clstrmgr`) is the core process in the HACMP for AIX system. The Cluster Manager, which runs on each node in the cluster, is essentially a monitoring and notification process. The Cluster Manager monitors cluster objects — node, network interface, and network — for changes in status. When a change occurs, the Cluster Manager generates an event that triggers the execution of an event script.

Solution

If the `maxperm` and `minperm` values are not tuned, a high amount of file page accesses could cause non-file pages to get paged out to paging space. This can include the working pages of the `clstrmgr` process itself.

This scenario can cause the `clstrmgr` process to be delayed when trying to reset its timer. As a result, the high amount of file accesses causes page-replacement activity, and one of the `clstrmgr` pages will be paged out. Due to the round-robin method of allocating paging space pages, this page can end up on one of the application volume group's paging space devices. If the application does a large number of writes to its filesystem, and the filesystem is on the same disk as that paging space, it could write a gigabyte of data. If the I/Os are random, the modifications stay in memory until a sync occurs. If the sync runs immediately after the write, this large amount of data gets flushed to the filesystem. If the `clstrmgr` then needs to run, it might need the page that was paged out. However, the page-in for this page can be blocked by the large amount of I/O going to the filesystem because they're both on the same disk. If it takes more than 5 seconds for the I/O to complete, the `clstrmgr` exceeds its default timeout value to reset the timer and causes the HACMP kernel extension to take over the node.

To prevent such a situation, configure the paging space so that there are no paging space volumes on disks where you expect a high amount of I/O activity.

Configuring a System That Has a Small Amount of Memory

Scenario

- You have a system that has very little memory and you're not able to add more memory.
- Your memory requirements are more than the amount of RAM on the system, and you expect a lot of paging to occur.

Solution

If there is only one paging space volume and if the system is paging heavily, the paging performance itself can suffer. Frames can be reused only after the page-outs are completed. However, if the frames will be occupied by pages that are currently residing in the paging space, the page-ins for those pages can't occur until the page-outs have completed. If the page-outs are scheduled to the same physical disk from which the page-ins are coming, then the page-ins are blocked.

If you have two paging space volumes on two different disks, the round-robin method of allocating paging space pages would cause the first four pages to get paged out to the first volume, and the next four to the second volume. While the second four pages are being written out, the first four might complete and page-ins can occur from that volume (assuming that the needed pages are on that volume). By having additional volumes on other disks, you can spread your I/Os across multiple disks.

Appendix A. Commands Available to Manage and Monitor Your Paging Space

Use the following commands to manage paging space:

chps Changes the attributes of a paging space.

lsps Displays the characteristics of a paging space.

mkps Adds an additional paging space.

rmps Removes an inactive paging space.

swapoff
Deactivates one or more paging space.

swapon
Activates a paging space.

Use the following commands to monitor paging space:

ps Shows current status of processes.

svmon
Captures and analyzes a snapshot of virtual memory.

vmstat
Reports virtual memory statistics.

Appendix B. Special Notices

This document was produced in the United States. IBM may not offer the products, programs, services or features discussed herein in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the products, programs, services, and features available in your area. Any reference to an IBM product, program, service or feature is not intended to state or imply that only IBM's product, program, service or feature may be used. Any functionally equivalent product, program, service or feature that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, service or feature.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed "AS IS". While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. The use of this information or the implementation of any techniques described herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. Customers attempting to adapt these techniques to their own environments do so at their own risk.

IBM is not responsible for printing errors in this publication that result in pricing or information inaccuracies.

The information contained in this document represents the current views of IBM on the issues discussed as of the date of publication. IBM cannot guarantee the accuracy of any information presented after the date of publication.

Any performance data contained in this document was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally available systems. Some measurements quoted in this document may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: AIX. A full list of U.S. trademarks owned by IBM can be found at <http://iplswww.nas.ibm.com/wpts/trademarks/trademar.htm>. Other company, product and service names may be trademarks or service marks of others.



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.