IBM Systems magazine

e-Newsletter Exclusive



# Paging, Memory and I/O Delays

## How to tune AIX versions 5.3, 6.1 and 7 for increased performance

September 2010 | by Jaqui Lynch

*Editor's Note: This is the first of a two-part series on AIX tuning. Part one covers paging, memory and I/O delays and part two will focus on disk I/O and the network.*

It's been a couple of years since I wrote an article on tuning AIX, so with the advent of AIX 7, this is a good time to review some of the basic tuning that may need to be done on your AIX systems. There have been many technology levels (TLs) released and some of the recommendations may have changed. In this article, I'll share AIX tuning information looking at tunables in AIX 5.3, 6.1 and 7.

I'll focus on the key areas of I/O, memory and network. By default AIX 6.1 and 7 do a fairly good job of tuning for memory and there are only a few small tweaks needed. AIX 5.3, however, needs a lot more tuning in this area. Figure 1 shows the different tunables with their default settings. The fourth column is the recommended value for those settings at the most recent TLs for all three versions.

One key point is: A fresh AIX 6 or 7 install will automatically install the new defaults for memory. If the system is migrated from AIX 5.3, then any tunables set in AIX 5.3 will be migrated across. Prior to performing a migration, it's suggested you make a note of all of the tunables that have been changed (take a copy of /etc/tunables/nextboot) and then reset the tunables to the defaults. After migration, check nextboot and make sure there's nothing in it. Now, go ahead and set the tunables that need to be changed for AIX 6 or 7.

### Page Spaces

Best practice suggests there be multiple paging spaces, all be equal in size and on different, non-busy hard disk drives (hdisks). All page spaces should either be mirrored or on a raided (1 or 5) storage area network (SAN). Unless your database requires it, there's normally no need to have twice the page space as you have memory. I have run large Oracle databases running AIX with 250 GB of memory and three 24 GB page spaces. The key is to use

technologies such as concurrent I/O (CIO) to avoid paging, as they have page spaces available in case you need them.

By default, AIX creates one page space (hd6) in rootvg that's too small. If rootvg is mirrored, then the page space will also be mirrored. I normally add a couple of custom-sized logical unit numbers (LUNs) from the SAN for the additional page spaces. Don't add extra page spaces to the same internal disks (or SAN LUNs) that your current rootvg page spaces are on. Having page spaces on the same hdisks can slow down paging.

When a virtual I/O server (VIOS) is first built, it's automatically configured with two page spaces, both on hdisk0. hd6 will be 512 MB and paging00 will be 1,024 MB. I always swap off paging00 and remove it and then increase hd6 to 4,096 MB. As mentioned, it's bad practice to have two page spaces share the same hdisk.

## Page Steal Method

The page_steal_method is set to 0 in AIX 5.3 default settings. This affects how the least recently used daemon (LRUDs) scan for pages to free. With lru_file_repage=0, a strong recommendation is made to the LRUD to favor executables and to always try to steal from filesystem (persistent) pages. It's far less expensive to steal from persistent pages than to steal from working storage pages, as the latter causes paging to/from page space. With 100 GB and five memory pools, memory is broken into five pools of roughly 20 GB a piece and each LRUD handles around 20 GB (this is a very simplistic view). Based on the numclient value in Figure 2, you can assume about 45 percent of our memory is being used for filesystems, which would be around 45 GB; the other 55 GB is working storage.

With page_steal_method=0, the LRUDs have to scan all of the memory pages they control when looking for free pages, even though it will most likely only free the persistent pages. Setting page_steal_method=1, the LRUDs move to a list-based page-management scheme. This means the LRUDs split memory into a list of persistent pages and a list of working storage pages. When the LRUDs are searching for pages to free from the filesystem cache, they will search only the persistent page lists. In the example in Figure 2, this should reduce the scan-to-free rate by more than a half, which will reduce overhead. The scan rate and free rate both show in the output from a "vmstat -I 2 2".

Figure 1

Figure 2

Figure 3

## Memory and I/O Buffers

When trying to determine the best settings for memory, there are several useful commands: in particular vmstat -v. Figure 2 shows a partial output from a vmstat -v.

There are two kinds of pages in memory: persistent pages, which are backed by filesystems, and working storage or dynamic pages, which include executables and their working areas. If a page from persistent is stolen, then it doesn't need to be paged out unless it was modified, in which case it gets written back to the filesystem. If a working storage page is stolen, it must first be written out to the page dataset and is then read back in from the page dataset when it's next needed; this is a far more expensive option in terms of performance.

Setting minperm%=3 with lru_file_repage=0 means a strong recommendation is made to the LRUD to always try to steal from persistent pages whenever filesystems are using more than 3 percent of memory. The LRUD will pretty much ignore the maximum settings with the exception that they can act as a cap for how much memory filesystems can use. As an example, maxperm% refers to all persistent pages—journaled file system (JFS), network file server (NFS), Veritas File System (VxFS) and enhanced jounaled file system (JFS2) and maxclient% is a subset thereof that refers to only NFS and JFS2 filesystems. maxperm% is a soft limit while maxclient% is a hard limit (and cannot exceed maxperm%). Since the new filesystems are normally JFS2, you should keep the maximums up at 90 percent so you don't accidentally cap the filesystems memory usage.

In the vmstat -v, there are several indicators to help you determine which values to tweak. In Figure 2, you can see numperm and numclient are identical, with both being 45.1 percent. This means 45.1 percent of memory is in use by NFS and/or JFS2 filesystems. If this is a database system, I would check to see if CIO is being used—as it can reduce both memory and CPU utilization by removing the double storage and handling of pages.

When an I/O is being built, the logical volume manager (LVM) requests a pbuf, which is a pinned memory buffer that holds the I/O request in the LVM layer. That I/O is then placed into another pinned memory buffer called an fsbuf. There are three types of fsbufs: filesystem fsbufs, which are for JFS filesystems; client fsbufs, which NFS and VxFS use; and external pager fsbufs, which JFS2 filesystems use. Additionally, there are psbufs, which are pinned memory buffers used for I/O requests to and from page space.

The values the vmstat-v command shows in Figure 2 are averages since boot. Since the server may not have been rebooted for a long time, it's important to take two snapshots a few hours apart to see if these numbers are changing. In this case, they were growing rapidly and required tuning changes.

Figure 1

Figure 2

Figure 3

## I/O Delays

In the vmstat -v output there are several indications of I/O delay, which will impact performance and memory. Below are some common ways to identify the causes of the blockages and to fix them.

1468217 pending disk I/Os blocked with no pbuf This line clearly indicates one or more pending disk I/Os are being blocked as they're trying to get pinned memory buffers (specifically pbufs). This indicates queuing at the LVM layer. It should be corrected using the lvmo command (see below), because AIX is unable to get a buffer to store information about a pending I/O request, causing the request to be delayed.

Figure 3 shows the output from the lvmo -a command, which indicates datavg has insufficient pbufs (look at the pervg_blocked_io_count). This should be corrected just for this one volume group using as these are pinned memory buffers and it does not make sense to oversize them:

```
lvmo -v datavg -o pv_pbuf_count=2048.
```

Normally I look at the current setting and, if it is 512 or 1024, I double it if it needs increasing.

11173706 paging space I/Os blocked with no psbuf The lvmo command also indicated some problems with pbufs in rootvg. After looking at the vmstat -v output, it's clear a large number of page space I/O requests have been blocked because no psbufs were available. psbufs are used to hold I/O requests at the virtual memory manager (VMM) layer and a lack of psbufs will adversely affect performance. It's also a sign that you're paging, which is a problem. The best way to fix these is to stop whatever is causing the paging. Another option is to add additional page spaces, as described earlier.

39943187 file system I/Os blocked with no fsbuf By default ,the system only provides 196 fsbufs for JFS filesystems. Prior to JFS2, these needed to be increased significantly (often to 2048) to ensure JFS I/O requests weren't being blocked in the filesystem layer due to a lack of fsbufs. Even when there's no JFS in the system, I sometimes see around 2,000 of these. However, the number above indicates the presence of a significant amount of blocked JFS I/Os in the system so, in this case, I would tune JFS and try and plan a move to JFS2—where you can take advantage of technologies like CIO for your databases. In AIX 6 and 7, numfsbufs is now a restricted parameter in the ioo command.

238 client file system I/Os blocked with no fsbuf Both NFS and VxFS are client filesystems and this line refers to the number of I/Os that were blocked at the filesystem layer due to a lack of client fsbufs. In order to fix this problem, further research will be needed to figure out whether this is a VxFS or an NFS problem.

1996487 external pager file system I/Os blocked with no fsbuf JFS2 is an external pager client filesystem and this line refers to the number of I/Os that were blocked at the

filesystem layer due to a lack of pinned memory fsbufs. Before, j2_nBufferPerPagerDevice was tuned to address this issue. JFS2 buffers are now corrected by increasing j2_dynamicBufferPreallocation using the ioo command. The default is 16 and I normally increase it slowly (try 32) as I try to reduce these I/O blockages.

## Other Memory

minfree and maxfree are other tunables in the vmo command that can impact paging. They're now allocated on a memory-pool basis and you can tell how many memory pools you have by issuing one of several commands. Depending on the version or the TL, you should get the information from either vmo -a, vmo -a -F (for 6.1) or vmstat -v.

If you overallocate these values, it's possible you'll see high values in the "fre" column of a vmstat and yet you'll be paging. The defaults are 960 and 1,088; I typically use 1,000 and 1,200 depending on other settings. The correct calculations for minfree and maxfree are dependant on the setting for j2MaxPageReadAhead, logical CPUs and memory pools.

In this case, let's say a vmstat shows there are 64 logical CPUs (lcpu) and 10 memory pools. Currently the defaults are set so minfree=960 and maxfree=1088. J2_maxPageReadahead=128. The calculation used for these current settings is:

```
M n=ma    0    20 lcpu  m mpool
Ma =m nfr      ma  ma pga  ad  2Ma  ag   ada  ad  lcpu  m mpool
```

There are 64 lcpu and 10 mempools. With j2_MaxPageReadahead =128 you get:

```
M n=ma    0    20  4   0  = ma    0   8 =  0
Ma =  0   ma  8  28   4   0  =   0 8  =  80
```

I would probably round this up to 2,048. Anytime j2_maxPageReadahead is modified, maxfree should be recalculated. So in this case, I'd leave minfree at 960 but would increase maxfree to 2,048.

## More Tuning Tips

Now, you should have a good handle on how to detect and address AIX performance issues related to paging, memory and I/O delays. Next month I'll focus on disk I/O and the network.