AIX and the Progress RDBMS

By Paul Koufalis

In the past 15 years, I have been fortunate enough to work with pretty much all the operating systems which Progress supported or supports. Does everyone remember VAX/VMS or the recently abandoned Dataserver for AS/400? I have to admit, though, that my heart only smiles for UNIX. There's just something about UNIX that appeals to the geek in me. Maybe it's the user-unfriendly character interface. Maybe it's vi (the best editor in the world!) or maybe it's just the fact that you can do whatever you want with it, good or bad. Regardless, Progress and UNIX are a match made in heaven.

Of course, the almost inevitable next question is: Which flavour of UNIX is best? The answer, also inevitably, is "It depends." Personally, I like them all. They're all quite similar but they all have their particularities, and that's what makes performance tuning so interesting. For this article, I decided to concentrate on IBM's AIX operating system, specifically versions 5.2 and above. I've also narrowed down my focus to two main categories: Memory and Disk I/O.

Memory

You can never have too much memory…unless you're running 32-bit Progress on AIX. That's when the dreaded 11 shared memory segment limit kicks in. I don't even think you can buy an AIX server with that little memory today! For those who are unfamiliar with this issue, 32-bit processes on AIX can only attach a maximum of 11 shared memory segments, each with a maximum size of 256 Mg. To make things worse, 32-bit Progress limits the maximum size of a shared memory segment to only 128 Mg. So effectively, the biggest –B with which you can start a database server is just under 1.4Gb, regardless of the available memory on the server. There are only two solutions to this problem: implement EXTSHM or upgrade to 64-bit OpenEdge 10.

I say EXTSHM is a solution but it's not one I personally would use. Even IBM discourages its use. From the man page of shmat():

*"The environment variable provides the option of executing an application either with the additional functionality of attaching more than 11 segments when **EXTSHM=ON**, or the **higher-performance access** to 11 or fewer segments when the environment variable is not set."*

There was also an issue in older versions of AIX whereby the entire server would crash if a process not having EXTSHM=on attempted to attach to EXTSHM shared memory segments. I do not know if this is still the case with AIX 5. Regardless, if you consider using EXTSHM make sure to test both the performance and stability implications extensively.

The real solution to memory limitations is to upgrade to the 64-bit version of OpenEdge 10. According to the 10.1A documentation, the new limit for –B is 116 Gb.

Virtual Memory

As of AIX 5.2, virtual memory kernel tuning parameters are set using vmo instead of vmtune, and the values are stored in the /etc/tunables directory. Typically, I only modify minperm%, maxperm%, and maxfree.

Minperm and maxperm loosely set the upper and lower limits of the page-replacement algorithm. According to the AIX documentation:

*"If percentage of RAM occupied by file pages rises above **maxperm**, page-replacement steals only file pages. If percentage of RAM occupied by file pages falls below **minperm**, page-replacement steals both file and computational pages. If percentage of RAM occupied by file pages is between **minperm** and **maxperm**, page-replacement steals only file pages unless the number of file repages is higher than the number of computational repages."*

The defaults for minperm and maxperm are 20% and 80% respectively. In simplified terms, this allows AIX to use up to 80% of RAM for file caching. In a database environment, we don't really need the Virtual Memory Manager (VMM) to cache pages for us because we're already caching our own pages in the database –B buffer cache. The VMM, while trying to help us avoid I/O by caching file pages is stealing memory we could use to increase –B! To make matters worse, if the amount of memory used is between minperm and maxperm, AIX may decide to page out –B pages instead of file pages! Imagine the effect that will have on database performance!

The solution is to drop minperm and maxperm to 5% and 10% respectively. That way, we're instructing the VMM to not use more than 10% of memory for file-caching. It may still use more than that prescribed 10% but only if no one else wants it. The actual amount of memory used is shown by the numperm parameter:

```
# vmstat -v
                  5.0 minperm percentage
                 10.0 maxperm percentage
                 40.4 numperm percentage
```

On this server, minperm and maxperm are correctly set to 5% and 10% but the VMM is still using 40% of available memory for file-caching. However it will immediately give it up if another process requests more memory. An interesting point: the excess of numperm over maxperm gives us a rough idea of how much more memory we could give to our Progress databases.

The third parameter that I normally change is maxfree. Minfree is the minimum number of free pages that must remain available. If the number of free pages drops below minfree, the page-stealing algorithm starts replenishing the free list. It stops when it hits maxfree. According to the AIX documentation, *"The difference between the **maxfree*** *and* ***minfree*** *parameters should always be equal to or greater than the value of the* ***maxpgahead*** *parameter, if you are using JFS."* I like to set maxpgahead to 256, and the default value of minfree is 960, therefore I increase maxfree to 1216. We'll discuss maxpgahead below in the disk I/O section.

Now, using vmstat or your favourite monitoring tool like nmon, you can see just how effectively your system is using memory:

```
--Memory-Use---------------------Paging------------------------Stats-----------
           Physical PagingSpace         pages/sec  In     Out  FileSystemCache
 % Used       99.3%       0.8%   to Paging Space   0.0    0.0  (numperm)  45.8%
 % Free        0.7%      99.2%   to File System 11597.1  157.0 Process    43.2%
 MB Used   31537.6MB     65.4MB  Page Scans        0.0         System     10.3%
 MB Free     206.4MB   8126.6MB  Page Cycles       0.0         Free        0.7%
 Total(MB) 31744.0MB   8192.0MB  Page Steals       0.0                   ------
                                 Page Faults    4424.8         Total     100.0%
 Min/Maxperm     1519MB(  5%)  3039MB( 10%) note: % of memory
 Min/Maxfree     960   1216      Total  Virtual   39.0GB       User       54.4%
 Min/Maxpgahead    2    256    Accessed Virtual   16.5GB 42.3% Pinned      8.0%
```

From this snapshot, I see a system that has 32Gb of physical memory. AIX doesn't like memory to sit idle so it's using 99.3% of it. It is not paging computational pages at all (which is very good) but it is reading pages from disk. If this snapshot is representative, numperm is 45.8% which tells me that the VMM is using 10Gb of memory that no one else wants (numperm% – maxperm% X total memory) and that I could put to better use.

Disk I/O

I can tell you from experience that if you don't set up your I/O subsystem correctly, it won't matter how fast your hardware is, how many gigs of write cache is installed, or what kind of fancy fibre controllers you have. Your I/O performance will be abysmal. And I'm not necessarily talking about RAID 5 vs. 10 here. What I'm talking about are some simple tweaks that can help you optimize your I/O throughput.

Logical Volume Setup

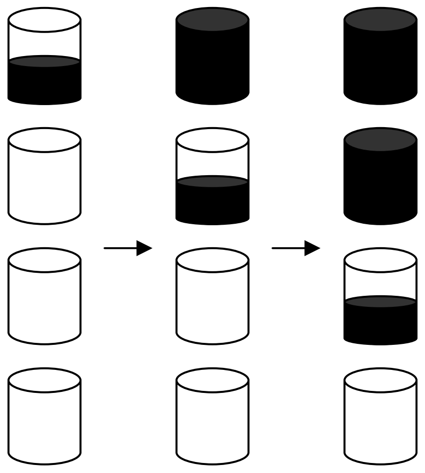
The AIX Logical Volume Manager (LVM) structures disk space in a three level hierarchy: The volume group, logical volume and file system. A volume group has one or more logical volumes, and each logical volume can have one filesystem. Physical disks are assigned to the volume group and disk space and location are assigned when creating the logical volume. There are some important points to consider when defining each of the levels.

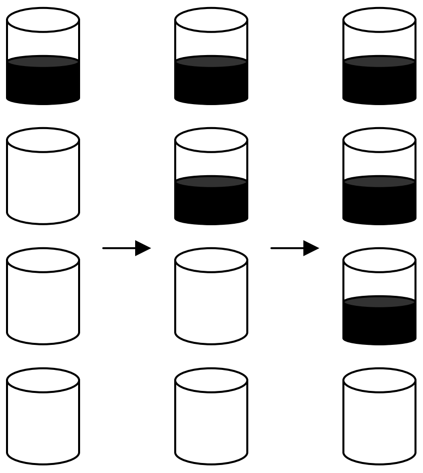
As mentioned above, physical volumes (PV's) are assigned to volume groups. These days, however, physical disks have been replaced by SAN "virtual" disks. These appear as PV's to the operating system but in reality are chunks created with the SAN's configuration tools. Setting up your SAN is far too big a topic to be discussed in a paragraph or two, however there is one important point: It is not a good idea to create one big multi-hundred gig virtual disk to present to the operating system. Instead, size the PV's such that it will take about 10 or 20 to fill your disk space needs. For example, if you need 500Gb, create 20 X 25Gb virtual disks on the SAN. The reason is simple but easily overlooked: disk throughput in AIX is affected by the number of hdisks/vpaths even if these hdisks are not real, physical hard drives but rather SAN representations. I'll discuss this bottleneck and others in much more detail below.

The next step is to create the logical volumes within the volume group. In the old days of real, physical, hard drives, we paid special attention to how we set up our logical volumes. RAID implementations were often done at the LV level. But with today's SAN disks, many people think that there is no need to tweak anything when creating the logical volume. Surprise, surprise…even with a fancy SAN in the background, the way you create your logical volumes will have a noticeable effect on your throughput.

First of all, you may consider striping your logical volume. At first glance, this seems futile as the PV is already striped within the SAN. But striping at the LVM level, in effect, *plaiding* the disk layout, increases disk throughput in benchmark tests. The downside to striping the LV comes when you try to add more disk space. The LVM doesn't like it when you try to add PV's to a striped logical volume.

An alternate solution is to set the *inter-physical volume allocation policy* to maximum. This effectively tells the LVM to use as many PV's as possible when creating the logical volume. This is effectively the same as striping, just with a much bigger stripe size. If you set the inter policy to minimum, data will be written to disk contiguously. But this is good, right? Well…not really.


Filling LV with inter-policy = min


Filling LV with inter-policy = max

By setting the inter-policy to minimum, you've effectively eliminated the advantages of striping. Even if the PV is physically striped on multiple SAN disks, the operating system will bottleneck as it attempts to execute all I/O operations on one hdisk. Iostat will show saturated I/O on that disk and virtually no I/O on the others.

Conclusion: *Plaiding* is fastest but harder to manage. Using an inter-policy of maximum is an excellent alternative.

JFS vs. JFS2

This is an easy one: Use JFS2 only on 64-bit kernels (not necessarily 64-bit Progress) and only on well patched versions of AIX 5.2 (ML6) and AIX 5.3 (ML3). There was a corruption bug in earlier versions of JFS2 that supposedly has been fixed in the above-mentioned ML levels. Myself, I'll wait a little longer…

As for JFS2 on 64-bit kernels only, this is an IBM directive that I have tested and confirmed. JFS2 throughput is inferior to JFS with the 32-bit kernel. My next step is to test inline logs versus traditionally logs. I have not compared the two yet but would love to hear from anyone out there that has done any testing.

I/O Tweaks

I'll finish off the disk I/O section with a list of parameters I like to tweak on AIX systems.

Maxpgahead = 256: This parameter tells the LVM how many pages to read ahead when doing sequential file access. Basically, if the LVM sees an application doing sequential file access, it jumps the gone and reads up to 256 pages into memory before the application actually asked for them.

Syncd interval = 10: By default, the sync daemon on AIX runs every 60 seconds. On a system with a high volume of writes, the spike can cause your application to freeze because Progress issues a sync call at the end of each checkpoint. Running the sync daemon more often increases the total I/O to disk but smoothes out the spikes. Note that if you're using –directio on your database, all your database changes will already be written to disk so the sync daemon will only have non-database modifications to write.

Sync_release_ilock = 1: This is an often-overlooked ioo parameter. By default, when the sync daemon writes a file's modifications to disk, it locks access to it via an i-node lock. Setting this value to one allows access to the file while its dirty pages are being written to disk.

Numfsbufs, pv_min_pbuf and the lvmo command: These are my favourite oft-overlooked ioo parameters. A quick peak at vmstat –v (or vmtune –a on AIX 5.1 or earlier) shows some interesting bottleneck information:

```
# vmstat –v
             <snip>
             54862015 pending disk I/Os blocked with no pbuf
                    0 paging space I/Os blocked with no psbuf
             55177977 filesystem I/Os blocked with no fsbuf
                    0 client filesystem I/Os blocked with no fsbuf
                    0 external pager filesystem I/Os blocked with no fsbuf
```

What are these blocked pending disk I/O's and filesystem I/O's? According to the AIX documentation:

"If there are many simultaneous or large I/Os to a filesystem or if there are large sequential I/Os to a file system, it is possible that the I/Os might bottleneck at the file system level while waiting for bufstructs."

The default value for numfsbufs is a ridiculously low 93 per filesystem. There's no magic number for this parameter so simply increase it slowly until the blocked filesystem I/O's disappear. I have had good results with numfsbufs at 4K or 8K.

Pv_min_pbuf defines the number of buffer structures created per physical volume. The default value of 512 is fine in most cases because a volume group will have many PV's. But if the volume group is created with only one big SAN hdisk, that 512 value quickly becomes inadequate. In case it isn't 100% clear, the "one big SAN disk" problem and the blocked pending disk I/O's bottleneck are one and the same. A real-life example:

```
# lsvg -p  dbvg
dbvg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk7            active            947         0           00..00..00..00..00

# lspv hdisk7
PHYSICAL VOLUME:    hdisk7                   VOLUME GROUP:     dbvg
PP SIZE:            256 megabyte(s)
TOTAL PPs:          947 (242432 megabytes)
```

The volume group "dbvg" above has only one big 240 Gb SAN disk. The resulting blocked I/O's are the ones shown in the vmstat –v a few paragraphs back.

Now let's look at another database volume group. This VG is also 240Gb but has 17 PV's:

```
# lsvg db01vg
VOLUME GROUP:       db01vg
VG STATE:           active        PP SIZE:        128 megabyte(s)
VG PERMISSION:      read/write     TOTAL PPs:      2023 (258944 megabytes)
TOTAL PVs:          17
```

I know for a fact that this second VG is much busier than the first, and yet vmstat –v shows almost no blocked I/Os:

```
# vmstat –v
             <snip>
              1340 pending disk I/Os blocked with no pbuf
                 0 paging space I/Os blocked with no psbuf
             40401 filesystem I/Os blocked with no fsbuf
                 0 client filesystem I/Os blocked with no fsbuf
                 0 external pager filesystem I/Os blocked with no fsbuf
```

An explanation can be found using the lvmo command:

```
# lvmo -a -v dbvg
vgname = dbvg
pv_pbuf_count = 512
total_vg_pbufs = 512
```

```
max_vg_pbuf_count = 16384
pervg_blocked_io_count = 55780490

# lvmo -a -v db01vg
vgname = db01vg
pv_pbuf_count = 512
total_vg_pbufs = 8704
max_vg_pbuf_count = 65536
pervg_blocked_io_count = 0
```

The total number of physical buffer structures for the entire volume group equals the per PV pbufs times the number of PV’s. In the first VG which has only one PV, the total pbufs for the entire VG = 512. In the second VG, that number is 17 X 512 = 8704.

If you do decide to change the value of pv_min_pbuf, be careful when setting it using the ioo command as the change would apply globally to all volume groups. Instead, consider setting it only for your database volume group via the lvmo command.

Conclusion

AIX is a superb platform for the Progress database, especially with the introduction of the 64-bit version of OpenEdge 10. The new pSeries hardware is insanely fast as are the DS-series SAN librairies. Hopefully the information in this article will help you get the most out of your AIX environment.

Paul Koufalis
pk@progresswiz.com

Paul began his Progress career in 1994 after finishing a computer engineering degree at McGill University. He started Progresswiz in 1999 and has been working as a Progress DBA, UNIX admin and all-round technical consultant ever since. If you’re in Vegas for Exchange 2006 be sure to sit in on his session: “OpenEdge Management in the Real World.”