Issue Date: IBM edition for Unix
April | May 2005, Posted On: 5/5/2005

Tuning AIX Commands for JFS2

Jaqui Lynch

Over the past few years many systems administrators have tuned their systems for journaled file systems (JFSs) by tweaking the minperm, maxperm, minpgahead, maxpgahead and maxrandwrt parameters. Moving forward, it's critical to revisit these parameters for a number of reasons, including that the vmtune and schedtune commands have been replaced in AIX V5.3. Any vmtune commands coded in /etc/inittab or an rc.local file are no longer being executed, as the command doesn't exist.

It's also important to understand how JFS parameters such as maxperm behave with JFS2, as they might have no negative impact until used on a JFS2 system.

Like most UNIX systems, AIX leaves in memory the pages—from local JFS, JFS2 or remote (NFS) file systems—of files that have been read or written. If they're referenced again, it saves an I/O operation, as they're already in memory. Since reads usually involve just a pagein—whereas computational- or working-set pages tend to involve both a pagein and a pageout—it's useful to tweak the minperm and maxperm values to favor computational pages. However, maxperm only affects JFS. JFS2's correct parameter is maxclient and many I/O problems that I look at involve misunderstandings about this parameter.

For JFS, maxperm specifies the file-page value (%) above which the page stealer should steal only file pages. The default is 80 percent, but this isn't a hard limit. If you set maxperm to 30 percent and the pages are available, I/O can still use up to 100 percent of memory. What you're changing is which pages get stolen when a page is needed. On the other end of the scale, minperm specifies the file-page value (%) below which the page stealer steals both file and computational pages. When the percentage of file pages falls between minperm and maxperm, the page stealer steals file pages unless the number of file repages exceeds the number of computational repages.

You can gain further control over the amount of memory used for caching client (NFS) and JFS2 pages with maxclient. It can't be set to a value greater than maxperm.

A key difference between maxperm and maxclient is how they're limited. When maxperm is set to 30 percent, it automatically caps maxclient at 30 percent, as maxclient can't exceed maxperm. However, maxperm is a soft limit controlled by strict_maxperm, which defaults to 0 or off. This means that file caching can use more memory than 30 percent, if it's available. Setting this low arbitrarily limits memory for JFS2 caching, since maxclient is a hard limit. The good news is that there's now a setting called strict_maxclient, which, when set to 0, makes the maxclient percentage a soft limit. I now automatically code this on JFS2 systems. In order to use strict_maxclient the system must be at v5.2 ML4 or higher.

Tuning the Virtual Machine Manager (VMM) write-behind parameters can provide additional performance benefits. When pages are updated in memory they aren't immediately written out to disk. Instead, dirty pages accumulate in memory until one of the following occurs:

- The syncd daemon runs (usually every 60 seconds)
- The number of free pages is reduced to minfree
- Someone issues a sync command
- A VMM write-behind threshold is reached

When the syncd daemon runs, it obtains and holds a lock on the i-node until all of the dirty pages have been written to disk. Any attempts to access that file will be blocked while the lock is held. On a busy system with a high I/O workload this can cause a lot of I/O wait and dramatically affect performance. There are three general ways to deal with this:

- Change syncd to run more often
- Set sync_release_ilock to 1, a potentially dangerous option that causes sync to flush all I/O to a file without holding the i-node lock and then use the i-node lock when it does the commit

- Activate random and/or sequential write-behind

I generally customize random write-behind and rarely, if ever, modify the syncd interval.

You can use numclust to control sequential write-behind. By default, files are partitioned into 16 K partitions or four pages; these are called clusters. If all four pages in a cluster are dirty, then the VMM will schedule the four dirty pages to go to disk when the first page in the next cluster is modified. The defaults are one cluster for JFS and eight clusters for enhanced JFS, but they can be increased to delay the writes. The JFS2 equivalent is j2_nRandomCluster, which defaults to 0. This option specifies the distance apart —in number of clusters—two consecutive writes must be in order to be considered random.

Random write-behind can dramatically affect performance on a system with a great deal of random I/O. If many pages have been modified, you'll see large bursts of I/O when the syncd daemon runs. This will affect performance consistency. You can use maxrandwrt to provide a threshold where dirty pages will be written out to disk. The default is 0 or never, but I tend to start with 32. This means that when 32 pages are dirty, any subsequent dirty pages will be written to disk. The initial 32 pages will be written out when the syncd daemon runs. For enhanced JFS, the equivalent is j2_maxRandomWrite, which also defaults to 0.

Finally, it's important to examine your file-system logging plan. By default, JFS filesystems use the one JFS log that's created by default for each volume group. In this case, I create a separate JFS log for each file system so that I have the flexibility to move that log if I get into performance conflicts where the log is fighting with the data.

JFS2's default is inline logging, where the log lives as part of the file system. You'll have performance problems with a high write workload if you use inline logs. And, in order to switch to external logs you must back up, delete and re-create the file system. I recommend creating external logs up front to give you the flexibility to move the logs and split the I/O load as needed. Once the logs are a separate physical entity, it's easy to move them live using mirroring techniques.

I normally use the following as a starting point for parameters:

```
vmo –p –o minperm%=10
vmo –p –o maxperm%=50           (I normally end up down at 30)
vmo –p –o maxclient%=50         (Must be <= maxperm%)
ioo   –p –o maxrandwrt=32
ioo   -p  -o j2_maxRandomWrite=32
vmo –p –o  strict_maxclient=0
```

Jaqui Lynch: *Jaqui Lynch, an eServer Magazine, IBM edition for UNIX technical editor, is a senior systems engineer focusing on pSeries and Linux at Mainline Information Systems. During her more than 26 years in the IS industry, she's been responsible for a wide variety of projects and OSs across multiple vendor platforms, including mainframes, UNIX systems, midrange systems and personal workstations. Jaqui can be reached at jaqui.lynch@mainline.com.*