Better Linux Disk Caching & Performance with vm.dirty_ratio & vm.dirty_background_ratio

Bob Plankers December 22, 2013

Best Practices, Cloud, System Administration, Virtualization

This is post #16 in my December 2013 series about Linux Virtual Machine Performance Tuning. For more, please see the tag "Linux VM Performance Tuning."

In previous posts on vm.swappiness and using RAM disks we talked about how the memory on a Linux guest is used for the OS itself (the kernel, buffers, etc.), applications, and also for file cache. File caching is an important performance improvement, and read caching is a clear win in most cases, balanced against applications using the RAM directly. Write caching is trickier. The Linux kernel stages disk writes into cache, and over time asynchronously flushes them to disk. This has a nice effect of speeding disk I/O but it is risky. When data isn't written to disk there is an increased chance of losing it.

There is also the chance that a lot of I/O will overwhelm the cache, too. Ever written a lot of data to disk all at once, and seen large pauses on the system while it tries to deal with all that data? Those pauses are a result of the cache deciding that there's too much data to be written asynchronously (as a non-blocking background operation, letting the application process continue), and switches to writing synchronously (blocking and making the process wait until the I/O is committed to disk). Of course, a filesystem also has to preserve write order, so when it starts writing synchronously it first has to destage the cache. Hence the long pause.

The nice thing is that these are controllable options, and based on your workloads & data you can decide how you want to set them up. Let's take a look:

```
$ sysctl -a | grep dirty
 vm.dirty_background_ratio = 10
 vm.dirty_background_bytes = 0
 vm.dirty_ratio = 20
 vm.dirty_bytes = 0
 vm.dirty_writeback_centisecs = 500
 vm.dirty_expire_centisecs = 3000
```

vm.dirty_background_ratio is the percentage of system memory that can be filled with "dirty" pages — memory pages that still need to be written to disk — before the pdflush/flush/kdmflush background processes kick in to write it to disk. My example is 10%, so if my virtual server has 32 GB of memory that's 3.2 GB of data that can be sitting in RAM before something is done.

vm.dirty_ratio is the absolute maximum amount of system memory that can be filled with dirty pages before everything must get committed to disk. When the system gets to this point all new I/O blocks until dirty pages have been written to disk. This is often the source of long I/O pauses, but is a safeguard against too much data being cached unsafely in memory.

vm.dirty_background_bytes and **vm.dirty_bytes** are another way to specify these parameters. If you set the _bytes version the _ratio version will become 0, and vice-versa.

vm.dirty_expire_centisecs is how long something can be in cache before it needs to be written. In this case it's 30 seconds. When the pdflush/flush/kdmflush processes kick in they will check to see how old a dirty page is, and if it's older than this value it'll be written asynchronously to disk. Since holding a dirty page in memory is unsafe this is also a safeguard against data loss.

vm.dirty_writeback_centisecs is how often the pdflush/flush/kdmflush processes wake up and check to see if work needs to be done.

You can also see statistics on the page cache in /proc/vmstat:

```
$ cat /proc/vmstat | egrep "dirty|writeback"
 nr_dirty 878
 nr_writeback 0
 nr_writeback_temp 0
```

In my case I have 878 dirty pages waiting to be written to disk.

Approach 1: Decreasing the Cache

As with most things in the computer world, how you adjust these depends on what you're trying to do. In many cases we have fast disk subsystems with their own big, battery-backed NVRAM

caches, so keeping things in the OS page cache is risky. Let's try to send I/O to the array in a more timely fashion and reduce the chance our local OS will, to borrow a phrase from the service industry, be "in the weeds." To do this we lower vm.dirty_background_ratio and vm.dirty_ratio by adding new numbers to /etc/sysctl.conf and reloading with "sysctl –p":

```
vm.dirty_background_ratio = 5
vm.dirty_ratio = 10
```

This is a typical approach on virtual machines, as well as Linux-based hypervisors. I wouldn't suggest setting these parameters to zero, as some background I/O is nice to decouple application performance from short periods of higher latency on your disk array & SAN ("spikes").

Approach 2: Increasing the Cache

There are scenarios where raising the cache dramatically has positive effects on performance. These situations are where the data contained on a Linux guest isn't critical and can be lost, and usually where an application is writing to the same files repeatedly or in repeatable bursts. In theory, by allowing more dirty pages to exist in memory you'll rewrite the same blocks over and over in cache, and just need to do one write every so often to the actual disk. To do this we raise the parameters:

```
vm.dirty_background_ratio = 50
vm.dirty_ratio = 80
```

Sometimes folks also increase the vm.dirty_expire_centisecs parameter to allow more time in cache. Beyond the increased risk of data loss, you also run the risk of long I/O pauses if that cache gets full and needs to destage, because on large VMs there will be a lot of data in cache.

Approach 3: Both Ways

There are also scenarios where a system has to deal with infrequent, bursty traffic to slow disk (batch jobs at the top of the hour, midnight, writing to an SD card on a Raspberry Pi, etc.). In that

case an approach might be to allow all that write I/O to be deposited in the cache so that the background flush operations can deal with it asynchronously over time:

```
vm.dirty_background_ratio = 5
vm.dirty_ratio = 80
```

Here the background processes will start writing right away when it hits that 5% ceiling but the system won't force synchronous I/O until it gets to 80% full. From there you just size your system RAM and vm.dirty_ratio to be able to consume all the written data. Again, there are tradeoffs with data consistency on disk, which translates into risk to data. Buy a UPS and make sure you can destage cache before the UPS runs out of power. :)

No matter the route you choose you should always be gathering hard data to support your changes and help you determine if you are improving things or making them worse. In this case you can get data from many different places, including the application itself, /proc/vmstat, /proc/meminfo, iostat, vmstat, and many of the things in /proc/sys/vm. Good luck!

Related

Adjust vm.swappiness to Avoid Unneeded Disk I/O
This is post #11 in my December 2013 series about Linux Virtual Machine Performance Tuning. For more, please see the tag "Linux VM Performance Tuning." The
December 11, 2013
In "Best Practices"

Use a RAM Disk to Improve Disk Access Times
This is post #15 in my December 2013 series about Linux Virtual Machine Performance Tuning. For more, please see the tag "Linux VM Performance Tuning." One of
December 14, 2013
In "Best Practices"

elevator=noop
I've also written about elevator=noop as part of my series on Linux performance tuning. The Linux kernel has various ways of optimizing disk I/O. One method it
February 21, 2008
In "Virtualization"

Comments on this entry are closed.