Greg Smith's Note Magnet

Sunday, August 3, 2008

A Linux write cache mystery

One happy coincidence for me last month is that I discovered a friend of mine had built a Linux-based server running PostgreSQL and put it into a colo, something I had been pricing out myself. He created me an account and now I've got a place to host some personal MediaWiki projects I'd been planning. One known issue my friend mentioned is that he'd been running into regular problems where the whole server just froze up for a few seconds. Because of the pattern of when it happened, he suspected it was related to heavy writes, and furthermore suspected the software RAID-1 implementation. Since that seemed unlikely to me, I told him to hold off on breaking the RAID until I could take a look at things.

The server is a quad-core system with 8GB of RAM and a pair of SATA disks in software RAID-1. The OS is CentOS 5.2, based on the RHEL5 release, and it's using kernel 2.6.18 (the regular one, not the Xen one).

I started by doing the same read/write testing I always do on a system whose I/O I don't necessarily trust or understand: that procedure is outlined at Testing your disks for PostgreSQL. Since the server has 8GB of RAM I used 2,000,000 blocks. I opened a couple of sessions to the server, executed that in one window, top in a second, and "vmstat 1" in a third. Watching a single second vmstat is one of the most useful things you can do for figuring out where bottlenecks are at on a system.

In this case, what quickly became apparent is that the system was alternating between healthy periods that looked like this:

```
procs -----------memory---------- ---swap-- -----io----
r  b   swpd   free   buff  cache   si   so    bi    bo
2  5   5196  46428  28160 6188640    0    0     0 53972(
0  6   5196  47268  28212 6047744    0    0     0 98840(
0  6   5196  46980  28272 6047216    0    0     0 64032
1  5   5196  45884  28336 6046788    0    0     0 61568
1  5   5196  47276  28400 6043408    0    0     0 65632
0  6   5196  46272  28460 6044080    0    0     0 65568
0  6   5196  48188  28524 6042420    0    0     0 65536
0  6   5196  46228  28592 6044836    0    0     0 66928
0  5   5196  46648  28652 6044812    0    0     0 61504
```

The bo (block out) number is the number to watch on this write test. That's in KB/s, so the entries in the later section here are all approximately 65MB/s. But at the beginning, it's writing in the Linux disk cache at a really high speed, as much as 988MB/s at the beginning. Note that these numbers are total I/O, which includes both of the disks in the RAID-1 pair. That means the actual per-disk write rate is closer to 32MB/s, a bit on the low side, but that's presumably because the disks are already backlogged with writes from the initial burst.

That's a healthy period. Here's what the unhealthy ones looked like:

```
procs -----------memory---------- ---swap-- -----io----
r  b   swpd   free   buff  cache   si   so    bi    bo
0  4   3780 935592  32048 5205528    0    0     0     0
0  4   3780 945140  32048 5205528    0    0     0     0
0  4   3780 954316  32048 5205528    0    0     0     0
0  4   3780 963616  32048 5205528    0    0     0     0
1  4   3780 973288  32048 5205528    0    0     0     0
0  4   3780 982464  32048 5205528    0    0     0     0
0  4   3780 992384  32048 5205528    0    0     0     0
0  4   3780 1002180  32048 5205528    0    0     0     (
0  4   3780 1011480  32048 5205528    0    0     0     (
0  4   3780 1021028  32048 5205528    0    0     0     (
0  4   3780 1030204  32048 5205528    0    0     0     (
0  4   3780 1039132  32048 5205528    0    0     0     (
0  4   3780 1049052  32048 5205528    0    0     0     (
```

That's over 20 seconds straight where zero blocks were written. That certainly seems to match the reported problem behavior of a long unresponsive period, and sure enough some of the sessions I had open were less responsive while this was going on. The question, then, is why it's happening? The disks seem to be working well enough; here's the summary at the end of the dd (the version of dd included in RHEL5 now provides this for you):

```
16384000000 bytes (16 GB) copied, 209.117 seconds, 78.3
```

78MB/s to each disk in the pair is completely reasonable.

I wrote a long paper on how Linux handles heavy writes called The Linux Page Cache and pdflush because I never found a source that really covered what happens in this situation. What I recommend there is watching /proc/meminfo to see what's going on. Here's a little shell bit you can execute to do that:

```
while [ 1 ]; do cat /proc/meminfo; sleep 1; done
```

With some practice you can note what numbers are constantly moving, catch when the bad behavior occurs, then hit control-C to break and peruse the last few entries in your terminal app

scrollback buffer. Here's what I found right around the same time as the lull periods:

```
MemTotal:      8174540 kB
MemFree:         62076 kB
Buffers:         21724 kB
Cached:        6158912 kB
SwapCached:          0 kB
Active:        1126936 kB
Inactive:      6101688 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:      8174540 kB
LowFree:         62076 kB
SwapTotal:    16771840 kB
SwapFree:     16766644 kB
Dirty:            6640 kB
Writeback:     3230792 kB ***
```

Note the line I starred there for Writeback. At the point where the system was stalling, a full 3.2GB of data was queued to write. That's 40% of RAM. Going back to my Linux page cache paper, you'll find that number listed: 40% is the point where Linux switches to the high dirty_ratio behavior, where all processes are blocked for writes. On a fast server with this much RAM, you can fill gigabytes of RAM in seconds, but writing that out to disk is still going to take a long time. If we work out the math, 3.2GB to write to two disks capable of 78MB/s each works out to...20.5 seconds. Well look at that, almost exactly the length of our long slow period, where process writes were stalled waiting for the kernel to clean up. I love it when the math comes together.

So, what to do? Well, this problem (way too many writes buffered on systems with large amounts of RAM) was so obvious that in the 2.6.22 Linux kernel, the defaults for the caching here were all lowered substantially. This is from the release notes to 2.6.22:

> Change default dirty-writeback limits. This means the kernel will write "dirty" caches differently...dirty_background_ratio defaulted to 10, now defaults to 5. dirty_ratio defaulted to 40, now it's 10

A check of this server showed it was using the 2.6.18 defaults as expected:

```
[gsmith@server ~]$ cat /proc/sys/vm/dirty_ratio
40
[gsmith@server ~]$ cat /proc/sys/vm/dirty_background_ra
10
```

So what I suggested to my friend the server admin was to change

these to the new values that are now standard in later kernels. It's easy to put these lines into /etc/rc.d/rc.local to make this change permanent after trying it out:

```
echo 10 > /proc/sys/vm/dirty_ratio
echo 5 > /proc/sys/vm/dirty_background_ratio
```

After doing that, I re-ran the dd test and things were much better. There were a few seconds where there was a small blip in throughput. During the 4 minute long test I found one 4-second long period writes dropped to the 2MB/s level. But for the most part, the giant bursts followed by lulls were gone, replaced by a fairly steady 130MB/s of writing the whole time. The final dd numbers looked like this after the retuning:

```
16384000000 bytes (16 GB) copied, 261.112 seconds, 62.7
```

So this did drop average and peak throughput a bit. That's usually how things worst: best throughput to disks usually involves writing in larger bursts, which is efficient but very disruptive. But that's a small price to pay for making the many second long pauses go away.

This particular problem shows up in all kinds of places where heavy writing is being done. Tuning these parameters is also one of the suggestions I make for people running PostgreSQL 8.2 or earlier in particular who want to tune checkpoint behavior better. In that context, there have even been reports of people turning this particular write cache off altogether, which again would lower average throughput, but in that case it was worth it for how much it decreased worst-case behavior.

Time will tell if there's anything else going on that was contributing to the pauses originally reported that is still happening on this server, but this looks like a pretty clear smoking gun that's now been holstered.
Posted by Greg Smith at 11:56 AM
Labels: linux, postgresql

13 comments:

tripy said...

Excellent writeup !
Memory management on linux has always been a bit of a "mystery box" to me.

I've learned a lot reading your post today.
Thank you very much!

August 3, 2008 at 2:05 PM

eggyknap said...

Some systems install watch(1) by default, which you can use instead of while [1] to reduce typing and highlight changes in output from one second to the next:

watch -n 1 -d "cat /proc/meminfo"

August 3, 2008 at 2:17 PM

chris said...

outstanding write up... This will definitely help those launching out on new projects.

I definitely have been exposed to some new things here.

August 3, 2008 at 10:12 PM

Greg Smith said...

"watch" is fun and useful for figuring out what's going on initially. The reason I didn't use it here is that I inevitably want to go back a bit after seeing something interesting, and the way it takes over the terminal isn't very compatible with using scrollback for that purpose. For example, when I was looking for the peak in Writeback, I waited until the value started going down rather than up, hit control-C, then went back a few seconds to find the exact peak.

Another command that is helpful to know about in this particular context is tee, which allows you to watch the activity on the screen and save it to a file. If you create a script and put something like this in it:

$ echo "while [1]; do cat /proc/meminfo; date; echo; sleep 1; done" > meminfo

$ chmod +x ./meminfo

$ ./meminfo | tee log

You then get that info on the screen and in a file you can review later with timestamps. tee is also handy for the vmstat output.

Now, if you have both watch and tee available, you can get really fancy like this:

watch -n 1 -d "(cat /proc/meminfo; date; echo) | tee -a log"

To get the best of both worlds--a full log and interactive delta tracking.

August 4, 2008 at 1:37 AM

RyanDBair said...

I had a very similar problem on an Intel based server. It turned out that due to a bug SAS controller firmware, the disk cache could be disabled. I updated the firmware, re-enabled the disk cache and its been smooth sailing ever since.

August 5, 2008 at 12:34 PM

Jeff said...

Great writeup! One question though:

If the disks are in RAID-1, it has to write the data to both disks anyway, so why are you counting the independent writing ability of the disks?

August 7, 2008 at 1:11 PM

Greg Smith said...

I'm not really counting them separately, I was just pointing out that the number reported by vmstat is a total of the two when showing that report.

August 10, 2008 at 5:30 PM

Bricklen said...

To highlight the Writeback column for easier recognition, you can also apply a bit of perl to your command:

while [1]; do cat /proc/meminfo | perl -p -e 's/(Writeback.*) /\033[46;1m$1\033[0m/ig;'; sleep 1; done

September 1, 2009 at 6:23 PM

Brian said...

while : ; do stuff ;done

October 1, 2009 at 12:49 PM

Brian said...

...or really, if the "stuff" is just one thing, or if one of the things in stuff is suitable, just use it directly in place of : or true or [1].

while cat /proc/meminfo ; sleep 1 ;done

use the cat as the testee, not the sleep, or some other thing that, if it should fail for some reason, will halt the loop instead of the loop trying to run something broken forever.

Apologies for focusing on such a nitpicky comparatively irrelevant issue compared to the gold in the actual meat of the article. I landed here because I was searching for insights of course, and here I found some good ones. Thanks much!

October 1, 2009 at 12:58 PM

pbsl said...
This comment has been removed by a blog administrator.
December 7, 2009 at 8:21 PM

BlackBurried said...

Why not just use O_DIRECT in your benchmarking and bypass VM cache altogether?

February 15, 2010 at 11:14 AM

Greg Smith said...

O_DIRECT will completely kill performance in lots of situations. You need the OS to do some caching to allow write combining, or any time you get two processes writing to two separate sections of disk you'll just seek between the two doing a single write at a time.

Also, O_DIRECT on Linux has been seriously buggy on many releases. Writes to the PostgreSQL WAL can be done in that mode, and we've gotten reports like PANIC caused by open_sync on Linux suggesting you can get both loud and undetected problems. One of the reasons Oracle, which relies on sync writes and O_DIRECT behavior more, is so careful about only certifying certain Linux kernels as usable is because their quality varies so much in this area. Seems like ext4 is finally doing all this the right way though as of 2.6.32, so maybe this will be more practical moving forward

February 16, 2010 at 6:01 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Blog Archive

► 2012 (1)
► 2011 (2)

About Me

Greg Smith

View my complete profile