How to avoid caching effects in read benchmarks

Asked 13 years, 3 months ago Modified 13 years, 3 months ago Viewed 3k times

4

I have a read benchmark and between consecutive runs, I have to make sure that the data does not reside in memory to avoid effects seen due to caching. So far what I used to do is: run a program that writes a large file between consecutive runs of the read benchmark. Something like

```
./read_benchmark
./write --size 64G --path /tmp/test.out
./read_benchmark
```

The write program simply writes an array of size 1G 64 times to file. Since the size of the main memory is 64G, I write a file that is approx. the same size. The problem is that writing takes a long time and I was wondering if there are better ways to do this, i.e. avoid effects seen when data is cached.

Also, what happens if I write data to /dev/null?

```
./write --size 64G --path /dev/null
```

This way, the write program exits very fast, no I/O is actually performed, but I am not sure if it overwrites 64G of main memory, which is what I ultimately want.

Your input is greatly appreciated.

c linux caching memory-management io

Share Improve this question Follow

asked Sep 15, 2011 at 19:44
Noz.i.kor
3,747 ●7 ●40 ●53

5 Answers

Sorted by: Highest score (default)

6

You can drop all caches using a special file in `/proc` like this:

```
echo 3 > /proc/sys/vm/drop_caches
```

That should make sure cache does not affect the benchmark.

Share Improve this answer Follow

answered Sep 15, 2011 at 19:47

Lukáš Lalinský
41.2k ● 6 ● 107 ● 128

2

You can just unmount the filesystem and mount it back. Unmounting flushes and drops the cache for the filesystem.

Share
Improve this answer
Follow

edited Sep 16, 2011 at 7:06

answered Sep 15, 2011 at 21:37

Maxim Egorushkin
136k ● 17 ● 194 ● 287

But that will still not clear a recently read file from the RAM, so I might see the effects of caching when I mount it again, wont I? – Noz.i.kor Sep 16, 2011 at 18:26

2 It clears the buffer cache of the device being unmounted. When it is mounted back the contents of the device are undetermined, so that the kernel can't assume the old cache is still valid (and the cache has been dropped at unmount anyway). – Maxim Egorushkin Sep 18, 2011 at 11:37

2

Use `echo 3 > /proc/sys/vm/drop_caches` to flush the pagecache, directory entries cache and inodes cache.

Share Improve this answer Follow

answered Sep 15, 2011 at 19:47

Arnaud Le Blanc
99.8k ● 24 ● 209 ● 196

I don't understand the distinction between main memory and cache? If I understand right, files are mapped to lazily loaded pages in the VM. If VM drops the page cache, what else is left? – HRJ Jul 20, 2013 at 7:30

2

You can the fadvise calls with FADV_DONTNEED to tell the kernel to keep certain files from being cached. You can also use mincore() to verify that the file is not cached. While the drop_caches solution is clearly simpler, this might be better than wiping out the entire cache as that effects all processes on the box.. I don't think you need elevated privledges to use fadvise while I bet you do for writing to /proc. Here is a good example of how to use fadvise calls for this purpose: http://insights.oetiker.ch/linux/fadvise/

Share Improve this answer Follow

answered Sep 15, 2011 at 20:05
frankc
11.5k ● 4 ● 33 ● 49

I am still not sure if the kernel will attempt to free pages from the main memory or just those from the processor cache. – Noz.i.kor Sep 15, 2011 at 20:30

I would be very surprised if it did not clear the main memory disk cache but you should be able to use mincore() to verify as it will tell you what pages of a file are in core – frankc Sep 15, 2011 at 21:15

2

One (crude) way that almost never fails is to simply occupy all that excess memory with another program.

Make a trivial program that allocates nearly all the free memory (while leaving enough for your benchmark app). Then `memset()` the memory to something to ensure that the OS will commit it to physical memory. Finally, do a `scanf()` to halt the program without terminating it.

By "hogging" all the excess memory, the OS won't be able to use it as cache. And this works in both Linux and Windows. Now you can proceed to do your I/O benchmark.

(Though this might not go well if you're sharing the machine with other users...)

Share
Improve this answer
Follow

edited Sep 15, 2011 at 20:49

answered Sep 15, 2011 at 19:47
Mysticial
471k ● 46 ● 339 ● 336

The key is to `memset()` to something to **ensure** that the OS will commit it to physical memory. I don't think a trivial `memset()` will ensure the commit, since the OS is most likely to delay executing this statement until it has to. From the perspective of the OS, if delaying or even not executing this statement won't cause any behavioral change, then it's a completely acceptable optimization. – Samuel Li Jan 13, 2019 at 20:25