Linux Page Cache Hit Ratio

31 Dec 2014

A recent Linux performance regression turned out to be caused by a difference in the page cache hit ratio: what was caching very well on the older system was caching poorly on the newer one. So how do you measure the page cache hit ratio directly?

How about a tool like this?:

```
# ./cachestat 1
Counting cache functions... Output every 1 seconds.
    HITS   MISSES  DIRTIES    RATIO   BUFFERS_MB   CACHE_MB
     210      869        0    19.5%            2        209
     444     1413        0    23.9%            8        210
     471     1399        0    25.2%           12        211
     403     1507        3    21.1%           18        211
     967     1853        3    34.3%           24        212
     422     1397        0    23.2%           30        212
[...]
```

This not only shows the size of the buffer and page cache, but also activity statistics. I've added cachestat to my perf-tools collection on github.

Longer Example

Here is some sample output followed by the workload that caused it:

```
# ./cachestat -t
Counting cache functions... Output every 1 seconds.
TIME         HITS   MISSES  DIRTIES    RATIO   BUFFERS_MB   CACHE_MB
08:28:57      415        0        0   100.0%            1        191
08:28:58      411        0        0   100.0%            1        191
08:28:59      362       97        0    78.9%            0          8
08:29:00      411        0        0   100.0%            0          9
08:29:01      775    20489        0     3.6%            0         89
08:29:02      411        0        0   100.0%            0         89
08:29:03     6069        0        0   100.0%            0         89
08:29:04    15249        0        0   100.0%            0         89
08:29:05      411        0        0   100.0%            0         89
08:29:06      411        0        0   100.0%            0         89
08:29:07      411        0        3   100.0%            0         89
[...]
```

I used the -t option to include the TIME column, to make describing the output easier.

The workload was:

```
# echo 1 > /proc/sys/vm/drop_caches; sleep 2; cksum 80m; sleep 2; cksum 80m
```

At 8:28:58, the page cache was dropped by the first command, which can be seen by the drop in size for "CACHE_MB" (page cache size) from 191 Mbytes to 8.

After a 2 second sleep, a cksum command was issued at 8:29:01, for an 80 Mbyte file (called "80m"), which caused a total of ~20,400 misses ("MISSES" column), and the page cache size to grow by 80 Mbytes. Each page is 4 Kbytes, so 20k x 4k == 80 Mbytes. The hit ratio during the uncached read dropped to 3.6%.

Finally, after another 2 second sleep, at 8:29:03 the cksum command was run a second time, this time hitting entirely from cache (the statistics spanning two output rows).

How It Works

I was curious to see whether ftrace, which is built into the Linux kernel, could measure cache activity, since ftrace function profiling provides efficient in-kernel counts. Systems can have a very high rate of cache activity, so we need to be careful to consider the overhead of any instrumentation.

While ftrace function profiling is cheap, its capabilities are also limited. It can count kernel function calls by-CPU, and show average latency. (It is the same facility used by funccount from perf-tools.) I can't, for example, use it with an advanced filter to match on function arguments or return values. It will only work if I deduce cache activity from the rate of kernel function calls alone.

For the kernels I'm studying (3.2 and 3.13), here are the four kernel functions I'm profiling to measure cache activity:

- mark_page_accessed() for measuring cache accesses
- mark_buffer_dirty() for measuring cache writes
- add_to_page_cache_lru() for measuring page additions
- account_page_dirtied() for measuring page dirties

mark_page_accessed() shows total cache accesses, and add_to_page_cache_lru() shows cache insertions (so does add_to_page_cache_locked(), which even includes a tracepoint, but doesn't fire on later kernels). I thought for a second that these two were sufficient: assuming insertions are misses, I have misses and total accesses, and can calculate hits.

The problem is that accesses and insertions also happens for writes, dirtying cache data. So the other two kernel functions help tease this apart (remember, I only have function call rates to work with here). mark_buffer_dirty() is used to see which of the accesses were for writes, and account_page_dirtied() to see which of the insertions were for writes.

It is possible that the kernel functions I'm using have been renamed (or are different logically) for your kernel version, and this script will not work as-is. I was hoping to use fewer than four functions, to make this more maintainable, but I didn't find a smaller set that worked for the workloads I tested.

If cachestat starts breaking too much for my kernel versions, I may rewrite it to use SystemTap or perf_events, which allow filtering.

Warnings

Instrumenting cache activity does cost some overhead, and this tool might slow your target system by 2% or so. Higher if you stress-test the cache. It also uses dynamic tracing of kernel functions, which could cause kernel freezes or panics, depending on your kernel version. Test before use.

The statistics should also be treated as best-effort. There may be some error margin depending on the frequency of unusual workload types, not properly matched by these four kernel functions. Test with a known workload to get confidence it will work for the intended target.

The Problem

When I encountered the earlier Linux performance regression, I didn't have cachestat. We had spotted a high rate of disk I/O, which led me to investigate the cause and work my way back to cache misses. I did this using custom ftrace and perf_events commands, measuring the rate of kernel functions and their call stacks.

While I got the job done, I wanted a better way for next time, which led to cachestat.

Other Techniques

I've found a few ways people commonly study the page cache hit ratio on Linux:

> A) Study the page cache miss rate by using iostat(1) to monitor disk reads, and assume these are cache misses, and not, for example, O_DIRECT. The miss rate is usually a more important metric than the ratio anyway, since misses are proportional to application pain. Also use free(1) to see the cache sizes.
>
> B) Drop the page cache (echo 1 > /proc/sys/vm/drop_caches), and measure how much performance gets worse! I love the use of a negative experiment, but this is of course a painful way to shed some light on cache usage.
>
> C) Use sar(1) and study minor and major faults. I don't think this works (eg, regular I/O).
>
> D) Use the cache-hit-rate.stp SystemTap script, which is number two in an Internet search for Linux page cache hit ratio. It instruments cache access high in the stack, in the VFS interface, so that reads to any file system or storage device can be seen. Cache misses are measured via their disk I/O. This also misses some workload types (some are mentioned in "Lessons" on that page), and calls ratios "rates".

I would have tried the SystemTap approach myself to begin with, but it can miss types including mmap'd reads and other kernel sources. For example, here's a call stack for mark_page_accessed() (a cache read), showing that we got here via a write() syscall:

```
           dd-30425 [000] 6788093.150288: mark_page_accessed: (mark_page_accessed+0x0/0x60)
           dd-30425 [000] 6788093.150291:
 => __getblk
 => __bread
 => ext3_get_branch
 => ext3_get_blocks_handle
 => ext3_get_block
 => __block_write_begin
 => ext3_write_begin
 => generic_perform_write
 => generic_file_buffered_write
 => __generic_file_aio_write
 => generic_file_aio_write
 => do_sync_write
 => vfs_write
 => sys_write
 => system_call_fastpath
```

It's reading file system metadata. This example uses ftrace, via my kprobe tool (perf-tools).

My preferred technique would be to modify the kernel to instrument page cache activity. Eg, either:

> E) Apply Keiichi's pagecache monitoring kernel patch, which provides tracepoints for cache instrumentation, and tools with awesome capabilities: not just system-wide ratios, but also per-process and per-file. I'd like this to be in mainline.
>
> F) Develop another kernel patch to add cache hit/miss statistics to /proc/meminfo.

And then, there's the approach I used myself for the issue: dynamic tracing of file system and disk I/O functions using ftrace and perf_events.

pcstat

If you're interested in page cache activity, you should also like pcstat, by Al Tobey, which uses mincore (or fincore), to see size how much files are present in the page cache. It's pretty awesome.

Conclusion

Hopefully in the future the kernel will provide an easy way to measure page cache activity, be it from /proc or tracepoints. In the meantime, I have cachestat which works for my kernel versions. Its current implementation is brittle, and may not work well on other versions without modifications, so its greatest value may be showing what can be done with a little effort.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).

About this blog

perf Examples
eBPF Tools
DTrace Tools
DTraceToolkit
DtkshDemos
Guessing Game
Specials
Books
Other Sites