BPF Use Case: Debugging Kernel Performance

Hao Luo, Chris Li

May 2023

Agenda

Introduction

Hao Luo: Share my experience of using BPF to root cause a kernel performance regression that happened in our production environment. Also raise some interesting questions.

Chris Li: Share our BPF tool that monitors swap faults and eliminated long tails of swap fault that happened in production.

Background

On 4.15 kernel, the p99 latency of writing cpuset.cpus is 7.5ms.

WriteResFileQuietOrDie | cpuset.cpus

Range	Bucket Size	Percent	Cumulative Percent
[0…1)	0	0.000%	0.000%
[1…1.5)	0	0.000%	0.000%
[1.5…2.25)	337402	44.155%	44.155%
[2.25…3.375)	294423	38.531%	82.686%
[3.375…5.0625)	103781	13.582%	96.268%
[5.0625…7.59375)	21690	2.839%	99.106%
[7.59375…11.3906)	5116	0.670%	99.776%
[11.3906…17.0859)	1197	0.157%	99.932%
[17.0859…25.6289)	212	0.028%	99.960%
[25.6289…38.4434)	88	0.012%	99.972%
[38.4434…57.665)	96	0.013%	99.984%
[57.665…86.4976)	88	0.012%	99.996%
[86.4976…129.746)	30	0.004%	100.000%
[129.746…194.62)	3	0.000%	100.000%

n=764126 *Sum*=2.08×10^6 *Mean*=2.729 *Aprx. median*=2.421 *Stddev*=1.638

Last updated: 0 seconds ago

Reset on: Mon, 06 Feb 2023 16:57:04 PST

4.15.0-smp-926.51.0.0
small_task_attractor
230 tasks

Background

On 5.10 kernel, the p99 latency of writing cpuset.cpus becomes around 86ms (11x longer tail latency)

WriteResFileQuietOrDie	cpuset.cpus	(histogram below)

Range	Bucket Size	Percent	Cumulative Percent
[0…1)	0	0.000%	0.000%
[1…1.5)	0	0.000%	0.000%
[1.5…2.25)	0	0.000%	0.000%
[2.25…3.375)	0	0.000%	0.000%
[3.375…5.0625)	0	0.000%	0.000%
[5.0625…7.59375)	0	0.000%	0.000%
[7.59375…11.3906)	0	0.000%	0.000%
[11.3906…17.0859)	0	0.000%	0.000%
[17.0859…25.6289)	0	0.000%	0.000%
[25.6289…38.4434)	1614	1.086%	1.086%
[38.4434…57.665)	60704	40.860%	41.946%
[57.665…86.4976)	74940	50.442%	92.388%
[86.4976…129.746)	10411	7.008%	99.396%
[129.746…194.62)	863	0.581%	99.976%
[194.62…291.929)	33	0.022%	99.999%
[291.929…437.894)	2	0.001%	100.000%

n=148567 *Sum*=9.36×10^6 *Mean*=63.03 *Aprx. median*=62.27 *Stddev*=16.49

Last updated: 0 seconds ago

Reset on: Wed, 08 Feb 2023 10:41:18 PST

5.10.0-smp-1100.402.0.0

small_task_attractor

230 tasks

Root Causing

BPF tool 1:

Let's check whether there is lock contention.

Methodology:

- Measure the time threads spent in the locking slow paths
  - Using a set of tracepoints in lock's slow paths (available since 5.19)
  - Profiling scoped within *cpuset_write_resmask()*
- Export the sampled time via ***BPF ringbuf*** and estimate quantiles
  - Alternative: BPF t-digest
- Compare the results on 5.10 against 4.15

Result:

- 5.10 does have much higher lock contention (under our production workloads)

Lock Contention Latency (unit: us)

	V4.15	V5.10
P25	0	2
P50	1	3
P75	1	10
P90	5	14607
P95	180	31886
P99	2589	111186

Root Causing

BPF tool 2:

Let's check what lock caused the contention.

Methodology 1:

- Perf lock contention can identify a few commonly used locks
- But not all locks can be identified, such as the locks in super_block (a super_block iterator may help).

Methodology 2:

- Attaches a program at the entry of contention, installs a timer of expiry time set to the tail latency (32ms as an example).
- When the timer fires, the callback records the lock's state (e.g. lock owner, lock address, etc)
    - Not all types of locks records owners unfortunately, but owner is a useful information.

Result:

- The lock of heavy contention is *cpuset_rwsem*

Root Causing

BPF tool 3:

Let's check *cpuset_rwsem*'s critical section.

Methodology:

- Check if there is significant increase of the critical sections of *cpuset_rwsem*.
- Attach programs at the acquisition and release of *cpuset_rwsem* and measure its time
- There are several non-overlaping functions that takes *cpuset_rwsem*.
  - Need to scope the profiling under them (I implemented a library that limits profiling scope)
  - Need to separate their profiled results.

Result:

- *Cpuset_write_resmask* has very long critical section

Root Causing

Examining the code path of cpuset_write_resmask(), identified the following callchain:

```
-> cpuset_write_resmask
  -> update_cpumask
    -> update_cpumasks_hier
      -> rebuild_sched_domains_locked
        -> partition_and_rebuild_sched_domains
            -> rebuild_root_domains
```

Rebuild_root_domains may be called within *cpuset_write_resmask* and it iterates all the cpuset cgroups and the tasks within the cgroup, which has very high overhead.

Fix: the iteration in rebuild_root_domains() is relevant to Deadline task accounting. We don't use DL tasks in our production, so we disabled it and report the problem to upstream.

With the fix, the p90 latency of writing cpuset.cpus is now around 17ms (much lower than before)
Note that there are still tails positioned at [17ms, 194ms].

WriteResFileQuietOrDie	cpuset.cpus

Range	Bucket Size	Percent	Cumulative Percent
[0…1)	0	0.000%	0.000%
[1…1.5)	0	0.000%	0.000%
[1.5…2.25)	584	50.607%	50.607%
[2.25…3.375)	425	36.828%	87.435%
[3.375…5.0625)	19	1.646%	89.081%
[5.0625…7.59375)	0	0.000%	89.081%
[7.59375…11.3906)	7	0.607%	89.688%
[11.3906…17.0859)	9	0.780%	90.468%
[17.0859…25.6289)	14	1.213%	91.681%
[25.6289…38.4434)	18	1.560%	93.241%
[38.4434…57.665)	16	1.386%	94.627%
[57.665…86.4976)	15	1.300%	95.927%
[86.4976…129.746)	17	1.473%	97.400%
[129.746…194.62)	26	2.253%	99.653%
[194.62…291.929)	3	0.260%	99.913%
[291.929…437.894)	1	0.087%	100.000%

n=1154 *Sum*=1.21×10^4 *Mean*=10.52 *Aprx. median*=2.241 *Stddev*=31.55
Last updated: 0 seconds ago
Reset on: Fri, 24 Feb 2023 13:33:00 PST

5.10 after fix

Further Optimization

We further identified that the use of *cpuset_rwsem* has high latency on locking for write (because of calling *synchronize_rcu*. *Can we detect that using BPF?*)

Switching *cpuset_rwsem* back to mutex eliminates those tails.

WriteResFileQuietOrDie	cpuset.cpus	(histogram below)

Range	Bucket Size	Percent	Cumulative Percent
[0…1)	0	0.000%	0.000%
[1…1.5)	0	0.000%	0.000%
[1.5…2.25)	7716	70.874%	70.874%
[2.25…3.375)	2784	25.572%	96.445%
[3.375…5.0625)	352	3.233%	99.679%
[5.0625…7.59375)	27	0.248%	99.927%
[7.59375…11.3906)	7	0.064%	99.991%
[11.3906…17.0859)	1	0.009%	100.000%

n=10887 $Sum=2.4\times10^4$ $Mean$=2.2 $Aprx.$ $median$=2.029 $Stddev$=0.5647

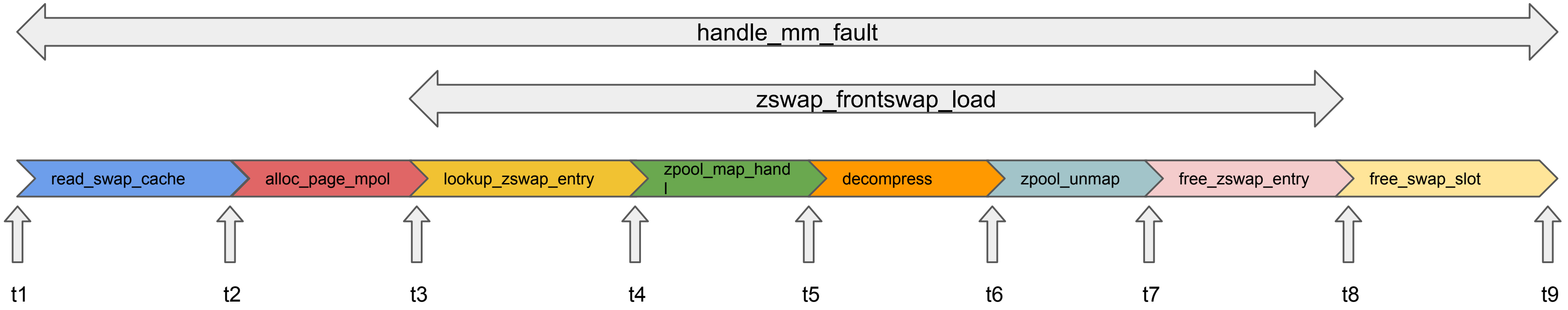
Last updated: 0 seconds ago

Reset on: Fri, 03 Mar 2023 15:24:16 PST

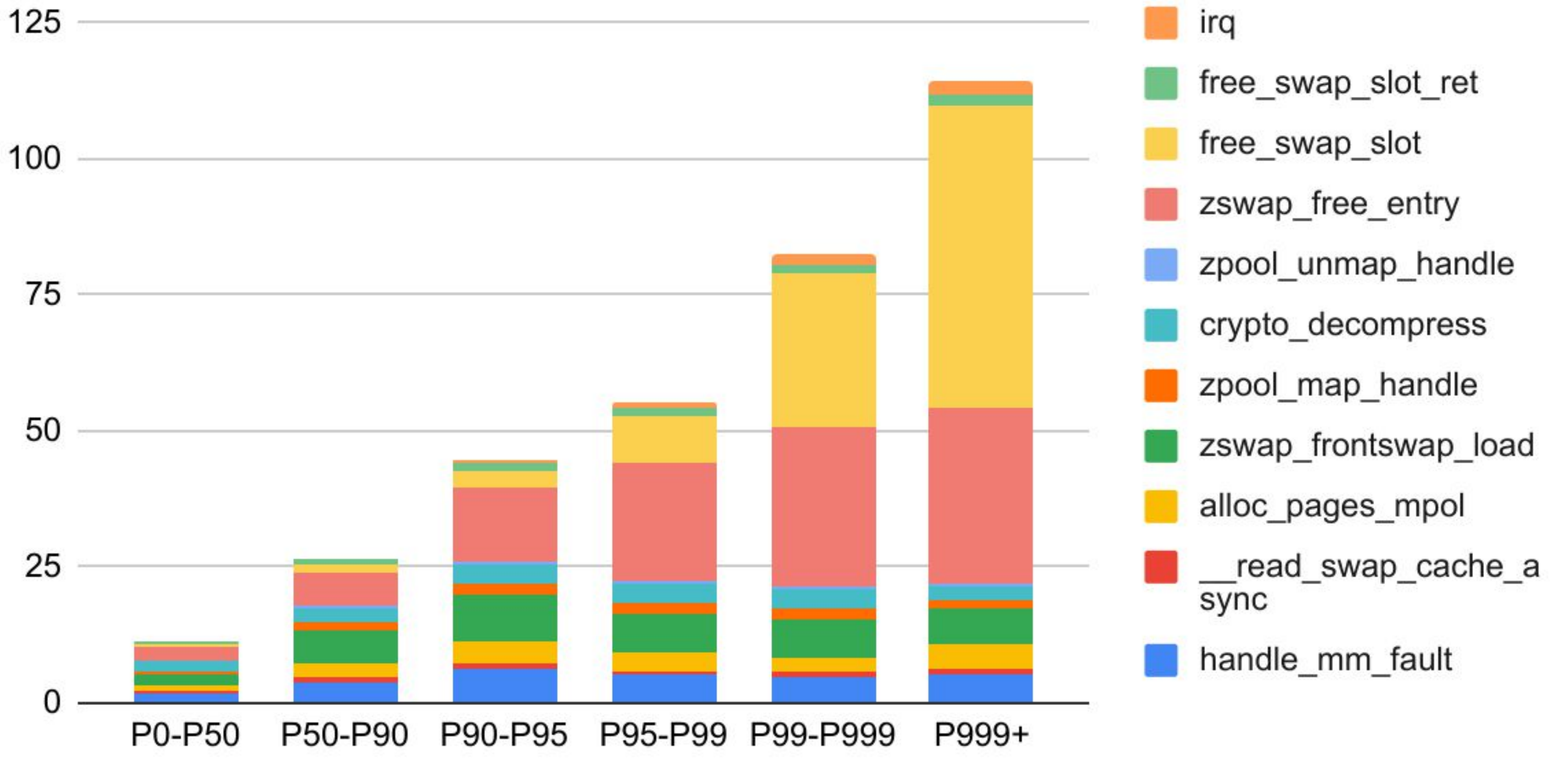
modified 5.10, cpuset_mutex

Swap Fault Latency Analysis

Zswap typically go through 8 function steps

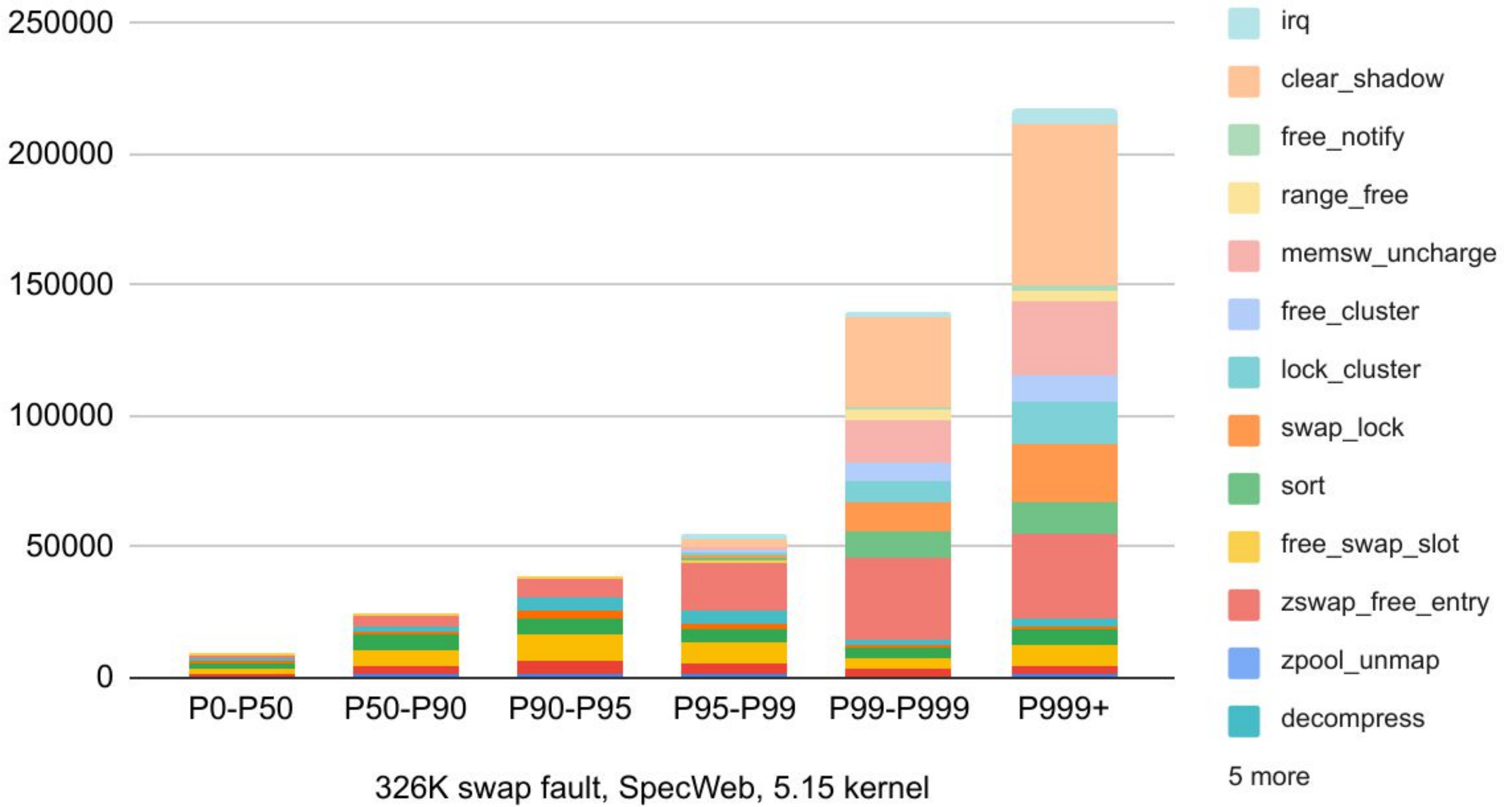


zswap fault latency (us) breakdown by percentile



38K sample SpecWeb with zswap, 4.15 kernel

zswap fault latency (ns) breakdown by percentile



326K swap fault, SpecWeb, 5.15 kernel

Q & A