September 14, 2022

cgroup rstat's advanced adoption

Michal Koutný, mkoutny@suse.com

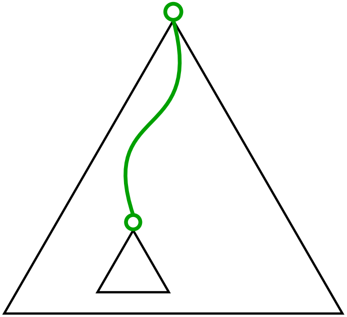
SUSE

cgroup recursive stats framework

- in this talk
  - give overview to the unfamiliar (to join discussion)
  - background for analysis
  - results of my experiments
  - ideas, next steps
- quick intro
  - `$cgroup_dir/*.stat`

Copyright © SUSE 2022

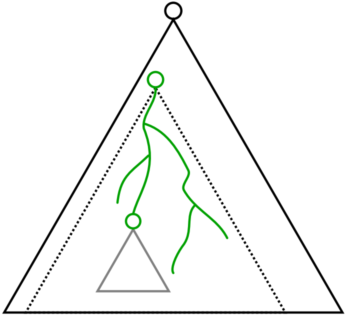
Update scope



- $O(\text{depth})$

Copyright © SUSE 2022

Flush scope



- $O(\text{subtree\_size} \cdot \text{nr\_controllers})$

Copyright © SUSE 2022

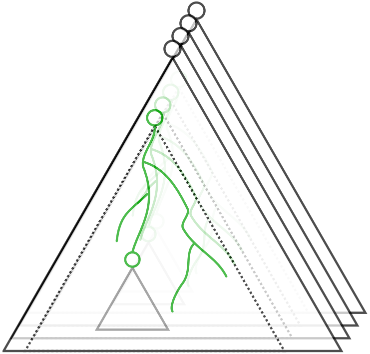
Update tree structure

```
                v---,
             .next -'  <---------,
             .children           |
    ,-----------'                |
   v                             |
.next -----> .next -----> .next -'
.children    .children    .children
   '--^         |   ^        '--^
                v   |
             .next -'
             .children
                '--^
```

- `struct cgroup_rstat_cpu`
- topological sort overall
- `.next` list is LIFO processed

Copyright © SUSE 2022

Full flush scope



- per-cpu replicas of the update tree

Copyright © SUSE 2022

Trade-offs and benefits

- updates are per-cpu
- flushing subtree of interest
- flushing subtree with updates

Copyright © SUSE 2022

Common updaters

- core: CPU time accounting (total and components)
- io controller: BIO dispatched
- memory controller: state, events, LRU stats
- BPF kfunc[1]

[1] bpf-next: a319185be9f5ad13c2a296d448ac52ffe45d194c

Copyright © SUSE 2022

Common flushers

- imprecise
  - `cpu.stat`
  - `io.stat`
  - `memory.stat`[2], `memory.zswap.current`
  - memcg: periodic flush (0.5 / s)
  - memcg: `workingset_refault`
- precise
  - memcg: `shrink_node` (reclaim)
  - memcg: `mem_cgroup_wb_stats` (writeback throttling)
  - BPF kfunc[3]
  - also cgroup removal

---

[2] Also memcg on v1

[3] bpf-next: a319185be9f5ad13c2a296d448ac52ffe45d194c

Copyright © SUSE 2022

Scalability

- locks
  - `cgroup_rstat_cpu_lock`, per-cpu spinlock
    - update traversal
    - per-cpu tree flushing
  - `cgroup_rstat_lock`
    - all flushing (possible release with `cond_resched`)
  - `stats_flush_lock` (memcg flush)
- individual controller flush callbacks
  - and time accounting in core
  - and BPF callbacks
- global accesses
  - `cgroup->self.parent` (update, flush)
  - `cgroup->rstat_css_list` (flush)
  - `cgroup_subsys_state->rstat_css_node` (flush)

Copyright © SUSE 2022

Memcg specialization

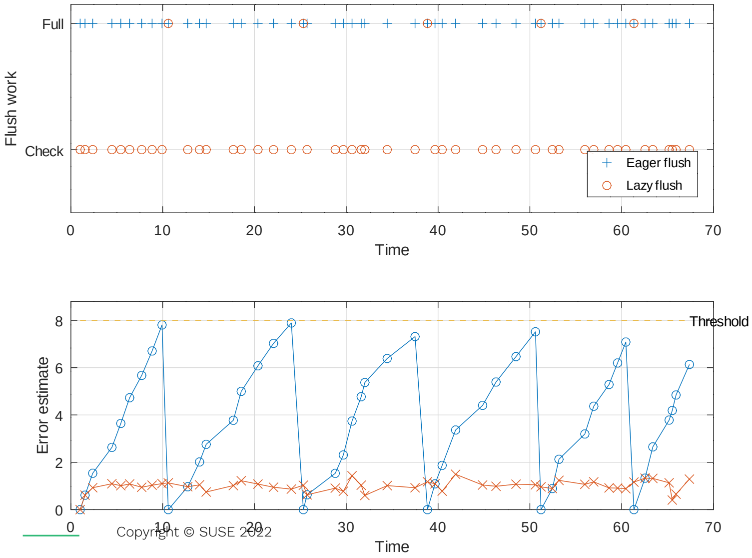
- expensive flush
  - `NR_VM_NODE_STAT_ITEMS` = 41
  - `NR_VM_EVENTS_ITEMS` = 107
  - `MEMCG_NR_STAT = NR_VM_NODE_STAT_ITEMS + 7`
- lazy flushing based on accumulated error estimate[4]
- error in $O(\text{nr\_cpus} \cdot \text{MEMCG\_CHARGE\_BATCH})$ or $O(\text{flush\_period} \cdot \text{cr}_{\text{max}})$

[4] error before rstat $O(\text{nr\_cpus} \cdot \text{MEMCG\_CHARGE\_BATCH} \cdot \text{subtree\_size})$ but no flushing work was necessary

Copyright © SUSE 2022

Copyright © SUSE 2022

Memcg specialization – theory



Copyright © SUSE 2022

Memcg specialization – theory

- average flush time in $O(\text{nr\_items}^2)$ [5] (strawman?)
    - full flush in $O(\text{nr\_items})$
    - frequency of reaching error estimate threshold in $O(\text{nr\_items})$

[5] 20220311160051.GA24796@blackbody.suse.cz

Copyright © SUSE 2022

Measurement setup

- 6.0.0-rc1 with a “masking” patch, 48 CPUs

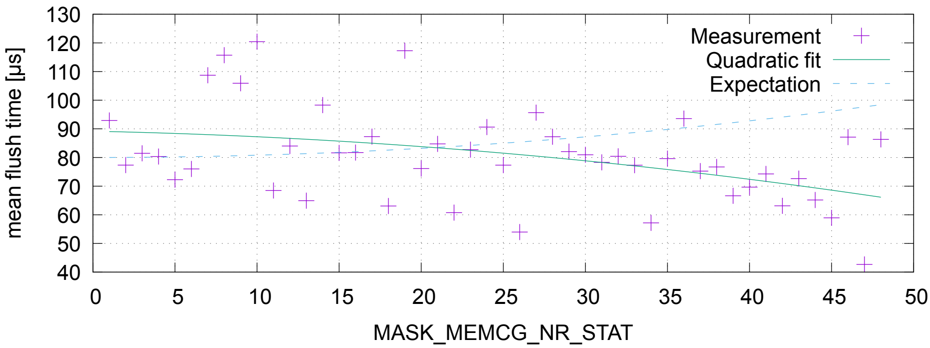
```
@@ -681,7 +685,8 @@ void __mod_memcg_state(
-        memcg_rstat_updated(memcg, val);
+        if (idx < MASK_MEMCG_NR_STAT)
+                memcg_rstat_updated(memcg, val);

@@ -5404,7 +5411,7 @@ static void mem_cgroup_css_rstat_flush(
-        for (i = 0; i < MEMCG_NR_STAT; i++) {
+        for (i = 0; i < MASK_MEMCG_NR_STAT; i++) {
```

- workload

```
systemd-run --scope -u measure.scope \
    -p MemoryMax=1600M \
    make -C $LINUX_TREE -j $NR_CPUS bzImage
```


Copyright © SUSE 2022

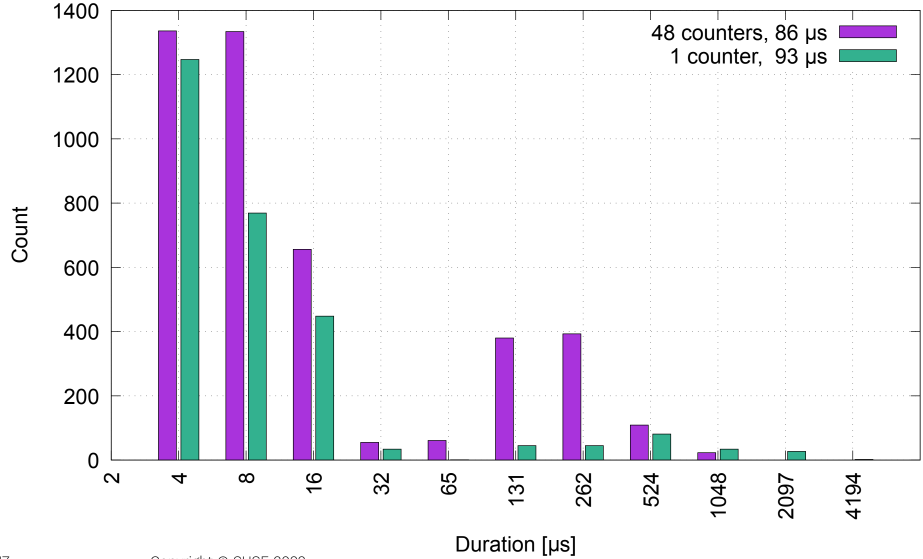
Measurement results



- basically noise

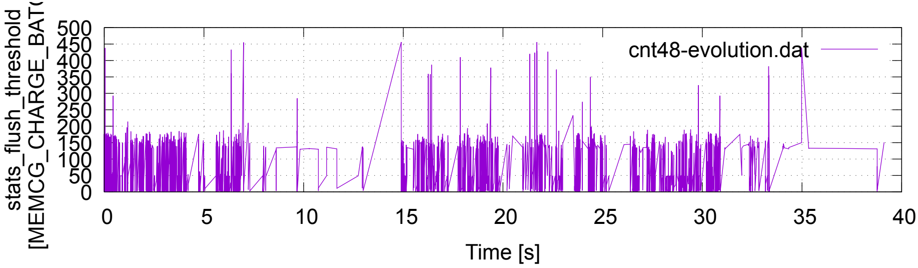
Copyright © SUSE 2022

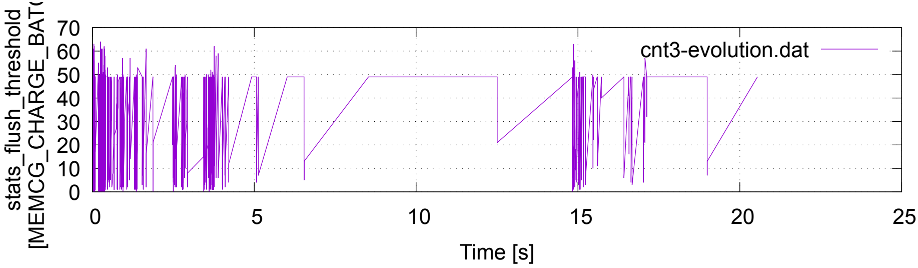
Measurement results 2



Copyright © SUSE 2022

Measurement results 3





Copyright © SUSE 2022

Copyright © SUSE 2022

Related work

- memcg: unify memcg stat flushing
  - piggy-back on running flushes in writeback calculation
  - bot: “we noticed a 47.8% improvement of fio.write_iops”
- [PATCH 3/3] memcg: reduce size of memcg vmstats structures
  - flushing less entries
- [PATCH v6 3/3] blk-cgroup: Optimize blkcg_rstat_flush()
  - flushing less devices

Copyright © SUSE 2022

Related work

- memcg: unify memcg stat flushing
  - piggy-back on running flushes in writeback calculation
  - bot: “we noticed a 47.8% improvement of fio.write_iops”
- [PATCH 3/3] memcg: reduce size of memcg vmstats structures
  - flushing less entries
- [PATCH v6 3/3] blk-cgroup: Optimize blkcg_rstat_flush()
  - flushing less devices
- cgroup: bpf: enable bpf programs to integrate with rstat
  - blank cheque in flush path
- [PATCH v7 1/4] mm: add NR_SECONDARY_PAGETABLE to count secondary page table uses.
  - yet another stat item

Copyright © SUSE 2022

Areas for improvement

- per controller update subtrees
  - smaller flushes (but less effective)
  - bigger memory footprint
- subtree flushing (memcg)
  - subtree locking?
  - harder error estimate tracking
- hardcode **`rstat_css_list`** iteration
- selective flush (required fields only)
- selective flush 2 (generalize memcg error-based flushing)
- detach rstatc from update tree *after* it's processed
  - frees time to writers

Copyright © SUSE 2022

The end

- Summary
  - The worry about $O(\text{nr\_items}^2)$ was not fulfilled.
  - Flushing still susceptible to take long (needs measurement).
- Thank you for attention!
- Q & A

Copyright © SUSE 2022