Recent changes in the kernel memory accounting

Roman Gushchin,

Facebook

2 years ago

Enabled by default on cgroup v1 and cgroup v2
Charging/uncharging is performed by the page allocator
Every charged page keeps a pointer (page->mem_cgroup) and a reference to the memory cgroup
Slab (SLAB/SLUB) infrastructure is replicated for each memory cgroup
Socket memory is an exception

Dying cgroups problem

Dying cgroup is a cgroup deleted by a user but pinned in the memory
Memory cgroup is a large objects (xxx KB or x MB)
The number of dying cgroups grew everywhere

Vmalloc-based kernel stacks

2 stacks are cached per CPU
Charging on allocation, uncharging on freeing
…
Or maybe not?
…
Charging on clone(), uncharging on exit()

VFS cache

Cgroups are created and destroyed
But some inodes and dentries stay
…
pinning original cgroups
How to release a memory cgroup without releasing all charged objects?

Slab reparenting

Recharge slab pages to the parent cgroup
How to do it efficiently?
…
page->mem_cgroup => slab_cache.memcg_params.memcg
All charges and statistics are fully recursive
slab_cache.memcg_params.memcg = parent_memcg
Merged into 5.3

Hm...

400k active task_structs?

Slab utilization problem

/proc/slab_info shows high 9x%, but it’s not true
If CONFIG_SLUB_CPU_PARTIAL is on
Real numbers were 15% to 65%
…
So is the memory overhead 0.2%?
cgroup.memory=nokmem saves ~50% of slab memory

New slab controller

Shared usage of slab caches and slab pages
Per-object tracking of slab objects
Reparenting

External memcg ownership data

```
#ifdef CONFIG_MEMCG
struct page {
        ...
        union {
                struct mem_cgroup *mem_cgroup;
                struct obj_cgroup **obj_cgroups;
        };
#endif
```

Byte-sized charging API & reparenting

```
struct obj_cgroup *get_obj_cgroup_from_current(void);
void obj_cgroup_get(struct obj_cgroup *objcg);
void obj_cgroup_put(struct obj_cgroup *objcg);
int obj_cgroup_charge(struct obj_cgroup *objcg, gfp_t gfp, size_t size);
void obj_cgroup_uncharge(struct obj_cgroup *objcg, size_t size);
```

Byte-sized statistics

NR_SLAB_RECLAIMABLE => NR_SLAB_RECLAIMABLE_B

NR_SLAB_UNRECLAIMABLE => NR_SLAB_UNRECLAIMABLE_B

Results

~40% memory savings with SLUB
~10+% memory savings with SLAB
xxx MB to x GB per host in Facebook's production
Reduced memory fragmentation
No known CPU regressions

Less (complicated) code

```
$ git diff --stat 85c250cafb31..6915d5907df3
 include/linux/memcontrol.h |  85 ++++++++++++++-
 mm/memcontrol.c            | 610 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++----------------------------------------
 mm/slab.h                  | 370 ++++++++++++++++++++++++++-------------------------------------
 mm/slab_common.c           | 643 +++-----------------------------------------------------------------------------------------------------------
 mm/slub.c                  | 229 ++++------------------------------------------
 mm/vmstat.c                |  30 +++++-
...
 21 files changed, 769 insertions(+), 1399 deletions(-) (without tests and tools)
```

Percpu memory accounting

Reuses the new slab controller design and code
Merged into 5.9
Memory cgroup internals are charged to the parent cgroup
TBD: percpu bpf maps (5.10?)

Kernel memory accounting now

- Significantly less expensive
- Less uniform
  - Not everything is handled by the page allocator
  - Per-page and per-object tracking
  - Memcg reference counting scheme is more complicated
  - Reparenting
- Better reflects different properties of different types of kernel memory
- Fewer gc issues

Thanks!

Vlastimil Babka
Jesper Dangaard Brouer
Shakeel Butt
Qian Cai
Nathan Chancellor
Mel Gorman
Tejun Heo
Michal Hocko
Naresh Kamboju
Michal Koutný
Christopher Lameter
Waiman Long
Chris Mason
Andrew Morton
Bharata B Rao
Mike Rapoport
Suleiman Souhlal
Johannes Weiner
Dennis Zhou