Contribution ID: **35** Type: **not specified**

Recent changes in the kernel memory accounting (or how to reduce the kernel memory footprint by ~40%)

Tuesday, 25 August 2020 10:00 (45 minutes)

Not a long time ago memcg accounting used the same approach for all types of pages.Each charged page had a pointer at the memory cgroup in the struct page. And it held a single reference to the memory cgroup, so that the memory cgroup structure was pinned in the memory by all charged pages.

This approach was simple and nice, but it didn't work well for some kernel objects,which are often shared between memory cgroups. E.g. an inode or a denty can outlive the original memory cgroup by far, because it can be actively used by someone else.Because there was no mechanism for the ownership change, the original memory cgroup was pinned in the memory, so that only a very heavy memory pressure could get rid of it.This lead to a so called dying memory cgroups problem: an accumulation of dying memory cgroups with uptime.

It has been solved by switching to an indirect scheme, where slab pages didn't reference the memory cgroup directly, but used a memcg pointer in the corresponding slab cache instead.The trick was that the pointer can be atomically swapped to the parent memory cgroup. In combination with slab caches reference counters it allowed to solve the dying memcg problem,but made the corresponding code even more complex: dynamic creation and destruction of per-memcg slab caches required a tricky coordination between multiple objects with different life cycles.

And the resulting approach still had a serious flow: each memory cgroup had it's own set of slab caches and corresponding slab pages. On a modern system with many memory cgroups it resulted in a poor slab utilization, which varied around 50% in my case. This made the accounting quite expensive: it almost doubled the kernel memory footprint.

To solve this problem the accounting has to be moved from a page level to an object level.If individual slab objects can be effectively accounted on individual level, there is no more need to create per-memcg slab caches. A single set of slab caches and slab pages can be used by all memory cgroups, which brings the slab utilization back to >90% and saves ~40% of total kernel memory.To keep the reparenting working and not reintroduce the dying memcg problem, an intermediate accounting vessel called obj_cgroup is introduced. Of course, some memory has to be used to store an objcg pointer for each slab object, but it's by far smaller than consequences of a poor slab utilization. The proposed new slab controller [1] implements a per-object accounting approach.It has been used on the Facebook production hosts for several months and brought significant memory savings (in a range of 1 GB per host and more) without any known regressions.

The object-level approach can be used to add an effective accounting of objects, which are by their nature not page-based: e.g. percpu memory. Each percpu allocation is scattered over multiple pages, but if it's small, it takes only a small portion of each page. Accounting such objects was nearly impossible on a per-page basis (duplicating chunk infrastructure will result in a terrible overhead),but with a per-object approach it's quite simple. Patchset [2] implements it. Perpcu memory is getting more and more used as a way to solve the contention problem on a multi-CPU system. Cgroups internals and bpf maps seem to be biggest users at this time, but likely new use cases will be added. It can easily take hundreds on MBs on a host, so if it's not account edit creates an issue in container memory isolation.

Links:
[1] https://lore.kernel.org/linux-mm/20200527223404.1008856-1-guro@fb.com/
[2] https://lore.kernel.org/linux-mm/20200528232508.1132382-1-guro@fb.com/

I agree to abide by the anti-harassment policy

I agree

Primary author: Mr GUSHCHIN, Roman (Facebook)

Presenter: Mr GUSHCHIN, Roman (Facebook)

Session Classification: LPC Refereed Track

Track Classification: LPC Refereed Track (Closed)