Subject: Re: [PATCH v3 16/28] memcg: kmem controller charge/uncharge infrastructure
Posted by Frederic Weisbecker on Wed, 30 May 2012 13:04:18 GMT
View Forum Message <> Reply to Message

On Fri, May 25, 2012 at 05:03:36PM +0400, Glauber Costa wrote:
> +struct kmem_cache *__mem_cgroup_get_kmem_cache(struct kmem_cache *cachep,
> + gfp_t gfp)
> +{
> + struct mem_cgroup *memcg;
> + int idx;
> + struct task_struct *p;
> +
> + gfp |= cachep->allocflags;
> +
> + if (cachep->memcg_params.memcg)
> + return cachep;
> +
> + idx = cachep->memcg_params.id;
> + VM_BUG_ON(idx == -1);
> +
> + p = rcu_dereference(current->mm->owner);
> + memcg = mem_cgroup_from_task(p);
> +
> + if (!mem_cgroup_kmem_enabled(memcg))
> + return cachep;
> +
> + if (memcg->slabs[idx] == NULL) {
> + memcg_create_cache_enqueue(memcg, cachep);
> + return cachep;
> + }
> +
> + return memcg->slabs[idx];
> +}
> +EXPORT_SYMBOL(__mem_cgroup_get_kmem_cache);
> +
> +bool __mem_cgroup_new_kmem_page(struct page *page, gfp_t gfp)
> +{
> + struct mem_cgroup *memcg;
> + struct page_cgroup *pc;
> + bool ret = true;
> + size_t size;
> + struct task_struct *p;
> +
> + if (!current->mm || in_interrupt())
> + return true;
> +
> + rcu_read_lock();

> + p = rcu_dereference(current->mm->owner);
> + memcg = mem_cgroup_from_task(p);
> +
> + if (!mem_cgroup_kmem_enabled(memcg))
> +  goto out;

Do you think it's possible that this memcg can be destroyed (like ss->destroy())
concurrently?

Probably not because there is a synchronize_rcu() in cgroup_diput() so as long
as we are in rcu_read_lock() we are fine.

OTOH current->mm->owner can exit() right after we fetched its memcg and thus the css_set
can be freed concurrently? And then the cgroup itself after we call rcu_read_unlock()
due to cgroup_diput().
And yet we are doing the mem_cgroup_get() below unconditionally assuming it's
always fine to get a reference to it.

May be I'm missing something?

> + mem_cgroup_get(memcg);
> +
> + size = (1 << compound_order(page)) << PAGE_SHIFT;
> +
> + ret = memcg_charge_kmem(memcg, gfp, size) == 0;
> + if (!ret) {
> +  mem_cgroup_put(memcg);
> +  goto out;
> + }
> +
> + pc = lookup_page_cgroup(page);
> + lock_page_cgroup(pc);
> + pc->mem_cgroup = memcg;
> + SetPageCgroupUsed(pc);
> + unlock_page_cgroup(pc);
> +
> +out:
> + rcu_read_unlock();
> + return ret;
> +}
> +EXPORT_SYMBOL(__mem_cgroup_new_kmem_page);