Subject: Re: [PATCH v2 06/11] memcg: kmem controller infrastructure
Posted by Glauber Costa on Wed, 15 Aug 2012 09:42:24 GMT
View Forum Message <> Reply to Message

>> + * memcg_kmem_new_page: verify if a new kmem allocation is allowed.
>> + * @gfp: the gfp allocation flags.
>> + * @handle: a pointer to the memcg this was charged against.
>> + * @order: allocation order.
>> + *
>> + * returns true if the memcg where the current task belongs can hold this
>> + * allocation.
>> + *
>> + * We return true automatically if this allocation is not to be accounted to
>> + * any memcg.
>> + */
>> +static __always_inline bool
>> +memcg_kmem_new_page(gfp_t gfp, void *handle, int order)
>> +{
>> + if (!memcg_kmem_on)
>> + return true;
>> + if (!(gfp & __GFP_KMEMCG) || (gfp & __GFP_NOFAIL))
>
> OK, I see the point behind __GFP_NOFAIL but it would deserve a comment
> or a mention in the changelog.

documentation can't hurt!

Just added.

> [...]
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index 54e93de..e9824c1 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
> [...]
>> +EXPORT_SYMBOL(__memcg_kmem_new_page);
>
> Why is this exported?
>

It shouldn't be. Removed.

>> +
>> +void __memcg_kmem_commit_page(struct page *page, void *handle, int order)
>> +{
>> + struct page_cgroup *pc;
>> + struct mem_cgroup *memcg = handle;
>> +

>> + if (!memcg)
>> + return;
>> +
>> + WARN_ON(mem_cgroup_is_root(memcg));
>> + /* The page allocation must have failed. Revert */
>> + if (!page) {
>> + size_t size = PAGE_SIZE << order;
>> +
>> + memcg_uncharge_kmem(memcg, size);
>> + mem_cgroup_put(memcg);
>> + return;
>> + }
>> +
>> + pc = lookup_page_cgroup(page);
>> + lock_page_cgroup(pc);
>> + pc->mem_cgroup = memcg;
>> + SetPageCgroupUsed(pc);
>
> Don't we need a write barrier before assigning memcg? Same as
> __mem_cgroup_commit_charge. This tests the Used bit always from within
> lock_page_cgroup so it should be safe but I am not 100% sure about the
> rest of the code.
>
Well, I don't see the reason, precisely because we'll always grab it
from within the locked region. That should ensure all the necessary
serialization.

>> +#ifdef CONFIG_MEMCG_KMEM
>> +int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta)
>> +{
>> + struct res_counter *fail_res;
>> + struct mem_cgroup *_memcg;
>> + int ret;
>> + bool may_oom;
>> + bool nofail = false;
>> +
>> + may_oom = (gfp & __GFP_WAIT) && (gfp & __GFP_FS) &&
>> +   !(gfp & __GFP_NORETRY);
>
> This deserves a comment.
>
can't hurt!! =)

>> +
>> + ret = 0;
>> +
>> + if (!memcg)
>> + return ret;

>> +
>> + _memcg = memcg;
>> + ret = __mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
>> +   &_memcg, may_oom);
>
> This is really dangerous because atomic allocation which seem to be
> possible could result in deadlocks because of the reclaim.

Can you elaborate on how this would happen?

> Also, as I
> have mentioned in the other email in this thread. Why should we reclaim
> just because of kernel allocation when we are not reclaiming any of it
> because shrink_slab is ignored in the memcg reclaim.

Don't get too distracted by the fact that shrink_slab is ignored. It is
temporary, and while this being ignored now leads to suboptimal
behavior, it will 1st, only affect its users, and 2nd, not be disastrous.

I see it this as more or less on pair with the soft limit reclaim
problem we had. It is not ideal, but it already provided functionality

>> +
>> + if (ret == -EINTR) {
>> + nofail = true;
>> + /*
>> +   * __mem_cgroup_try_charge() chosed to bypass to root due to
>> +   * OOM kill or fatal signal. Since our only options are to
>> +   * either fail the allocation or charge it to this cgroup, do
>> +   * it as a temporary condition. But we can't fail. From a
>> +   * kmem/slab perspective, the cache has already been selected,
>> +   * by mem_cgroup_get_kmem_cache(), so it is too late to change
>> +   * our minds
>> +   */
>> + res_counter_charge_nofail(&memcg->res, delta, &fail_res);
>> + if (do_swap_account)
>> +   res_counter_charge_nofail(&memcg->memsw, delta,
>> +   &fail_res);
>
> Hmmm, this is kind of ugly but I guess unvoidable with the current
> implementation. Oh well...
>

Oh well...