Subject: Re: [PATCH v5 06/14] memcg: kmem controller infrastructure
Posted by Glauber Costa on Thu, 18 Oct 2012 09:16:19 GMT
View Forum Message <> Reply to Message

On 10/18/2012 02:12 AM, Andrew Morton wrote:
> On Tue, 16 Oct 2012 14:16:43 +0400
> Glauber Costa <glommer@parallels.com> wrote:
>
>> This patch introduces infrastructure for tracking kernel memory pages to
>> a given memcg. This will happen whenever the caller includes the flag
>> __GFP_KMEMCG flag, and the task belong to a memcg other than the root.
>>
>> In memcontrol.h those functions are wrapped in inline acessors. The
>> idea is to later on, patch those with static branches, so we don't incur
>> any overhead when no mem cgroups with limited kmem are being used.
>>
>> Users of this functionality shall interact with the memcg core code
>> through the following functions:
>>
>> memcg_kmem_newpage_charge: will return true if the group can handle the
>> allocation. At this point, struct page is not
>> yet allocated.
>>
>> memcg_kmem_commit_charge: will either revert the charge, if struct page
>> allocation failed, or embed memcg information
>> into page_cgroup.
>>
>> memcg_kmem_uncharge_page: called at free time, will revert the charge.
>>
>> ...
>>
>> +static __always_inline bool
>> +memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup **memcg, int order)
>> +{
>> + if (!memcg_kmem_enabled())
>> + return true;
>> +
>> + /*
>> + * __GFP_NOFAIL allocations will move on even if charging is not
>> + * possible. Therefore we don't even try, and have this allocation
>> + * unaccounted. We could in theory charge it with
>> + * res_counter_charge_nofail, but we hope those allocations are rare,
>> + * and won't be worth the trouble.
>> + */
>> + if (!(gfp & __GFP_KMEMCG) || (gfp & __GFP_NOFAIL))
>> + return true;
>> + if (in_interrupt() || (!current->mm) || (current->flags & PF_KTHREAD))
>> + return true;

```
>> +
>> + /* If the test is dying, just let it go. */
>> +        if (unlikely(test_thread_flag(TIF_MEMDIE)
>> +                     || fatal_signal_pending(current)))
>> +  return true;
>> +
>> + return __memcg_kmem_newpage_charge(gfp, memcg, order);
>> +}
>
> That's a big function!  Why was it __always_inline?  I'd have thought
> it would be better to move the code after memcg_kmem_enabled() out of
> line.
>
```

it is big, but it is mostly bit testing. So the goal here is to avoid a
function call at all costs, this being a fast path.

> Do we actually need to test PF_KTHREAD when current->mm == NULL?
> Perhaps because of aio threads whcih temporarily adopt a userspace mm?

I believe so. I remember I discussed this in the past with David
Rientjes and he advised me to test for both.

```
>
>> +/**
>> + * memcg_kmem_uncharge_page: uncharge pages from memcg
>> + * @page: pointer to struct page being freed
>> + * @order: allocation order.
>> + *
>> + * there is no need to specify memcg here, since it is embedded in page_cgroup
>> + */
>> +static __always_inline void
>> +memcg_kmem_uncharge_page(struct page *page, int order)
>> +{
>> + if (memcg_kmem_enabled())
>> +  __memcg_kmem_uncharge_page(page, order);
>> +}
>> +
>> +/**
>> + * memcg_kmem_commit_charge: embeds correct memcg in a page
>> + * @page: pointer to struct page recently allocated
>> + * @memcg: the memcg structure we charged against
>> + * @order: allocation order.
>> + *
>> + * Needs to be called after memcg_kmem_newpage_charge, regardless of success or
>> + * failure of the allocation. if @page is NULL, this function will revert the
>> + * charges. Otherwise, it will commit the memcg given by @memcg to the
>> + * corresponding page_cgroup.
```

>> + */
>> +static __always_inline void
>> +memcg_kmem_commit_charge(struct page *page, struct mem_cgroup *memcg, int order)
>> +{
>> + if (memcg_kmem_enabled() && memcg)
>> + __memcg_kmem_commit_charge(page, memcg, order);
>> +}
>
> I suspect the __always_inline's here are to do with static branch
> trickery. A code comment is warranted if so?
>

Not necessarily. Same thing as above. We want to avoid function calls in
those sites.
