Subject: Re: [PATCH 07/10] memcg: Stop res_counter underflows.
Posted by Glauber Costa on Wed, 29 Feb 2012 17:05:52 GMT
View Forum Message <> Reply to Message

On 02/28/2012 08:07 PM, Suleiman Souhlal wrote:
> On Tue, Feb 28, 2012 at 5:31 AM, Glauber Costa<glommer@parallels.com> wrote:
>> I don't fully understand this.
>> To me, the whole purpose of having a cache tied to a memcg, is that we know
>> all allocations from that particular cache should be billed to a specific
>> memcg. So after a cache is created, and has an assigned memcg,
>> what's the point in bypassing it to root?
>>
>> It smells like you're just using this to circumvent something...
>
> In the vast majority of the cases, we will be able to account to the cgroup.
> However, there are cases when __mem_cgroup_try_charge() is not able to
> do so, like when the task is being killed.
> When this happens, the allocation will not get accounted to the
> cgroup, but the slab accounting code will still think the page belongs
> to the memcg's kmem_cache.
> So, when we go to free the page, we assume that the page belongs to
> the memcg and uncharge it, even though it was never charged to us in
> the first place.
>
> This is the situation this patch is trying to address, by keeping a
> counter of how much memory has been bypassed like this, and uncharging
> from the root if we have any outstanding bypassed memory.
>
> Does that make sense?
>
Yes, but how about the following:

I had a similar problem in tcp accounting, and solved that by adding
res_counter_charge_nofail().

I actually implemented something very similar to your bypass (now that I
understand it better...) and gave up in favor of this.

The tcp code has its particularities, but still, that could work okay
for the general slab.

Reason being:

Consider you have a limit of X, and is currently at X-1. You bypassed a
page.

So in reality, you should fail the next allocation, but you will not -
(unless you start considering the bypassed memory at allocation time as

well).

If you use res_counter_charge_nofail(), you will:

1) Still proceed with the allocations that shouldn't fail - so no
   difference here
2) fail the normal allocations if you have "bypassed" memory filling
   up your limit
3) all that without coupling something alien to the res_counter API.

Subject: Re: [PATCH 07/10] memcg: Stop res_counter underflows.
Posted by Suleiman Souhlal on Wed, 29 Feb 2012 19:17:27 GMT
View Forum Message <> Reply to Message

On Wed, Feb 29, 2012 at 9:05 AM, Glauber Costa <glommer@parallels.com> wrote:
> On 02/28/2012 08:07 PM, Suleiman Souhlal wrote:
>>
>> On Tue, Feb 28, 2012 at 5:31 AM, Glauber Costa<glommer@parallels.com>
>> wrote:
>>>
>>> I don't fully understand this.
>>> To me, the whole purpose of having a cache tied to a memcg, is that we
>>> know
>>> all allocations from that particular cache should be billed to a specific
>>> memcg. So after a cache is created, and has an assigned memcg,
>>> what's the point in bypassing it to root?
>>>
>>> It smells like you're just using this to circumvent something...
>>
>>
>> In the vast majority of the cases, we will be able to account to the
>> cgroup.
>> However, there are cases when __mem_cgroup_try_charge() is not able to
>> do so, like when the task is being killed.
>> When this happens, the allocation will not get accounted to the
>> cgroup, but the slab accounting code will still think the page belongs
>> to the memcg's kmem_cache.
>> So, when we go to free the page, we assume that the page belongs to
>> the memcg and uncharge it, even though it was never charged to us in
>> the first place.
>>
>> This is the situation this patch is trying to address, by keeping a
>> counter of how much memory has been bypassed like this, and uncharging
>> from the root if we have any outstanding bypassed memory.
>>
>> Does that make sense?
>>

> Yes, but how about the following:
>
> I had a similar problem in tcp accounting, and solved that by adding
> res_counter_charge_nofail().
>
> I actually implemented something very similar to your bypass (now that I
> understand it better...) and gave up in favor of this.
>
> The tcp code has its particularities, but still, that could work okay for
> the general slab.
>
> Reason being:
>
> Consider you have a limit of X, and is currently at X-1. You bypassed a
> page.
>
> So in reality, you should fail the next allocation, but you will not -
> (unless you start considering the bypassed memory at allocation time as
> well).
>
> If you use res_counter_charge_nofail(), you will:
>
> 1) Still proceed with the allocations that shouldn't fail - so no
>   difference here
> 2) fail the normal allocations if you have "bypassed" memory filling
>   up your limit
> 3) all that without coupling something alien to the res_counter API.

Ok. I'll give it a try.

Thanks!
-- Suleiman
