Subject: Re: [PATCH 00/10] memcg: Kernel Memory Accounting.
Posted by Glauber Costa on Wed, 29 Feb 2012 16:47:36 GMT
View Forum Message <> Reply to Message

On 02/28/2012 07:47 PM, Suleiman Souhlal wrote:
> Hello,
>
> On Tue, Feb 28, 2012 at 5:03 AM, Glauber Costa<glommer@parallels.com>  wrote:
>> Hi,
>>
>>
>> On 02/27/2012 07:58 PM, Suleiman Souhlal wrote:
>>>
>>> This patch series introduces kernel memory accounting to memcg.
>>> It currently only accounts for slab.
>>>
>>> It's very similar to the patchset I sent back in October, but
>>> with a number of fixes and improvements.
>>> There is also overlap with Glauber's patchset, but I decided to
>>> send this because there are some pretty significant differences.
>>
>>
>> Indeed. I will comment in details at your patches. I hope we can reach an
>> agreement soon about the best way to move forward.
>
> Thanks a lot.
>
>>> Slab gets accounted per-page, by using per-cgroup kmem_caches that
>>> get created the first time an allocation of that type is done by
>>> that cgroup.
>>
>>
>> I don't particularly care, but doesn't that introduce potential high
>> latencies for task within the cgroup?
>>
>> Having the cgroup creation taking longer to complete is one thing,
>> but having an object allocation taking a lot longer than it would
>> take otherwise can be detrimental to some users. Again, for my usecase, this
>> is not terribly important.
>
> Given that this is only done for the first allocation of a particular
> type in a cgroup, and that cache creation shouldn't be *that*
> expensive, I don't really think this is a big concern, unless your
> cgroups are *extremely* short-lived.
>
> That said, if you really think this is a problem, it would be trivial
> to change the code to always  do the creation of the per-cgroup
> kmem_caches asynchronously.

Well, it should simplify the code a bit, and if we always lose one
object, it also shouldn't be that bad, right?

> We already to do this when the first allocation is not GFP_KERNEL: We
> let the allocation use the regular cache (so it won't be accounted to
> the cgroup), and defer the creation to a workqueue.
>
>> Maybe we can pick a couple of caches that we know to be of more importance,
>> (like the dcache), and create them at memcg create time ?
>
> I don't understand. Are you suggesting pre-allocating some caches,
> like dcache, at cgroup creation, and letting "less important" caches
> get created on-demand?
> I guess that would be possible, but I'm not sure it's really
> necessary, given what I mentioned above.

Ok. It is just crossed my mind, but I don't feel strongly. Kind of thing
we can defer to when and if someone complains.

>>> The main difference with Glauber's patches is here: We try to
>>> track all the slab allocations, while Glauber only tracks ones
>>> that are explicitly marked.
>>> We feel that it's important to track everything, because there
>>> are a lot of different slab allocations that may use significant
>>> amounts of memory, that we may not know of ahead of time.
>>
>>
>> Note that "explicitly marking" does not mean not tracking them all in the
>> end. I am just quite worried about having the heavy-caches like the dcache
>> lying around without a proper memcg-shrinker.
>
> I agree that we definitely need memcg-aware shrinkers.
> We have patches for those, but I didn't want to include them in this
> patchset, so that we could concentrate on getting the accounting right
> first.
> Once we can reach consensus on how to do the accounting, I will be
> happy to start discussing shrinking. :-)
>
>>> A per-cgroup kmem_cache will appear in slabinfo named like its
>>> original cache, with the cgroup's name in parenthesis.
>>
>>
>> There is another problem associated with names. You can use just the
>> trailing piece of the cgroup name, since it may lead to duplicates like
>> memory/foo and memory/bar/foo all named "dcache-foo" (which I believe is
>> what you do here, right?). But then you need to include the full path, and
>> can end up with some quite monstruous things that will sure render

>> /proc/slabinfo useless for human readers.
>
> I hadn't considered the fact that two cgroups could have the same base name.
> I think having a name makes it a lot easier for a human to understand
> which cgroup is using slab, so what about having both the base name of
> the cgroup AND its css id, so that the caches are named like
> "dentry(5:foo)"?

That would be better, so if you really want to keep names, I'd advise
you to go this route.

However, what does "5" really means to whoever is reading that? css ids
are not visible to the user, so there isn't really too much information
you can extract for that. So why not only dentry-5 ?

> This should let us use the name of the cgroup while still being able
> to distiguish cgroups that have the same base name.

I am fine with name+css_id if you really want names on it.

>> I was thinking: How about we don't bother to show them at all, and instead,
>> show a proc-like file inside the cgroup with information about that cgroup?
>
> One of the patches in the series adds a per-memcg memory.kmem.slabinfo.
I know. What I was wondering was if we wanted to show only the
non-cgroup slabs in /proc/slabinfo, and then show the per-cgroup slabs
in the cgroup only.

>
>>>  - There is a difference with non-memcg slab allocation in
>>>  that with this, some slab allocations might fail when
>>>  they never failed before. For example, a GFP_NOIO slab
>>>  allocation wouldn't fail, but now it might.
>>>  We have a change that makes slab accounting behave
>>>  the same as non-accounted allocations, but I wasn't sure
>>>  how important it was to include.
>>
>> How about we don't account them, then ?
>
> The patch we have (but I haven't included) makes it so that we account
> for them most of the times. However, when we are in OOM conditions,
> they get bypassed to the root (or not accounted for at all, if that's
> what you prefer).
> I guess I will make sure the patch is included in the v2 of this series.

I don't like this bypass thing. But I'll comment in details on the patch
that adds it now.