Subject: Re: [PATCH 00/23] slab+slub accounting for memcg
Posted by Suleiman Souhlal on Mon, 30 Apr 2012 21:43:28 GMT
View Forum Message <> Reply to Message

On Fri, Apr 20, 2012 at 2:48 PM, Glauber Costa <glommer@parallels.com> wrote:
> Hi,
>
> This is my current attempt at getting the kmem controller
> into a mergeable state. IMHO, all the important bits are there, and it should't
> change *that* much from now on. I am, however, expecting at least a couple more
> interactions before we sort all the edges out.

Thanks a lot for doing this.

> This series works for both the slub and the slab. One of my main goals was to
> make sure that the interfaces we are creating actually makes sense for both
> allocators.
>
> I did some adaptations to the slab-specific patches, but the bulk of it
> comes from Suleiman's patches. I did the best to use his patches
> as-is where possible so to keep authorship information. When not possible,
> I tried to be fair and quote it in the commit message.
>
> In this series, all existing caches are created per-memcg after its first hit.
> The main reason is, during discussions in the memory summit we came into
> agreement that the fragmentation problems that could arise from creating all
> of them are mitigated by the typically small quantity of caches in the system
> (order of a few megabytes total for sparsely used caches).
> The lazy creation from Suleiman is kept, although a bit modified. For instance,
> I now use a locked scheme instead of cmpxcgh to make sure cache creation won't
> fail due to duplicates, which simplifies things by quite a bit.

I actually noticed that, at least for slab, the cmpxchg could never
fail due to kmem_cache_create() already making sure that duplicate
caches could not be created at the same time, while holding
cache_mutex_mutex.

I do like your simplification though.

>
> The slub is a bit more complex than what I came up with in my slub-only
> series. The reason is we did not need to use the cache-selection logic
> in the allocator itself - it was done by the cache users. But since now
> we are lazy creating all caches, this is simply no longer doable.
>
> I am leaving destruction of caches out of the series, although most
> of the infrastructure for that is here, since we did it in earlier
> series. This is basically because right now Kame is reworking it for

> user memcg, and I like the new proposed behavior a lot more. We all seemed
> to have agreed that reclaim is an interesting problem by itself, and
> is not included in this already too complicated series. Please note
> that this is still marked as experimental, so we have so room. A proper
> shrinker implementation is a hard requirement to take the kmem controller
> out of the experimental state.

We will have to be careful for cache destruction.
I found several races between allocation and destruction, in my patchset.

I think we should consider doing the uncharging of kmem when
destroying a memcg in mem_cgroup_destroy() instead of in
pre_destroy(), because it's still possible that there are threads in
the cgroup while pre_destroy() is being called (or for threads to be
moved into the cgroup).

-- Suleiman

---