Subject: Re: [PATCH v2 06/11] memcg: kmem controller infrastructure
Posted by Greg Thelen on Tue, 14 Aug 2012 18:58:10 GMT
View Forum Message <> Reply to Message

On Mon, Aug 13 2012, Glauber Costa wrote:

>>> > + WARN_ON(mem_cgroup_is_root(memcg));
>>> > + size = (1 << order) << PAGE_SHIFT;
>>> > + memcg_uncharge_kmem(memcg, size);
>>> > + mem_cgroup_put(memcg);
>> Why do we need ref-counting here ? kmem res_counter cannot work as
>> reference ?
> This is of course the pair of the mem_cgroup_get() you commented on
> earlier. If we need one, we need the other. If we don't need one, we
> don't need the other =)
>
> The guarantee we're trying to give here is that the memcg structure will
> stay around while there are dangling charges to kmem, that we decided
> not to move (remember: moving it for the stack is simple, for the slab
> is very complicated and ill-defined, and I believe it is better to treat
> all kmem equally here)

By keeping memcg structures hanging around until the last referring kmem page is uncharged do such zombie memcg each consume a css_id and thus put pressure on the 64k css_id space? I imagine in pathological cases this would prevent creation of new cgroups until these zombies are dereferenced.

Is there any way to see how much kmem such zombie memcg are consuming? I think we could find these with for_each_mem_cgroup_tree(root_mem_cgroup). Basically, I'm wanting to know where kernel memory has been allocated. For live memcg, an admin can cat memory.kmem.usage_in_bytes. But for zombie memcg, I'm not sure how to get this info. It looks like the root_mem_cgroup memory.kmem.usage_in_bytes is not hierarchically charged.