Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure
Posted by Michal Hocko on Fri, 05 Oct 2012 13:47:23 GMT
View Forum Message <> Reply to Message

On Thu 04-10-12 07:43:16, Tejun Heo wrote:
[...]
> > That is right but I think that the current discussion shows that a mixed
> > (kmem disabled and kmem enabled hierarchies) workloads are far from
> > being theoretical and a global knob is just too coarse. I am afraid we
>
> I'm not sure there's much evidence in this thread. The strongest upto
> this point seems to be performance overhead / difficulty of general
> enough implementation. As for "trusted" workload, what are the
> inherent benefits of trusting if you don't have to?

One advantage is that you do _not have_ to consider kernel memory
allocations (which are inherently bound to the kernel version) so the
sizing is much easier and version independent. If you set a limit to XY
because you know what you are doing you certainly do not want to regress
(e.g. because of unnecessary reclaim) just because a certain kernel
allocation got bigger, right?

> > will see "we want that per hierarchy" requests shortly and that would
> > just add a new confusion where global knob would complicate it
> > considerably (do we really want on/off/per_hierarchy global knob?).
>
> Hmmm? The global knob is just the same per_hierarchy knob at the
> root. It's hierarchical after all.

When you said global knob I imagined mount or boot option. If you want
to have yet another memory.enable_kmem then IMHO it is much easier to
use set-accounted semantic (which is hierarchical as well).

> Anyways, as long as the "we silently ignore what happened before being
> enabled" is gone, I won't fight this anymore. It isn't broken after
> all.

OK, it is good that we settled this.

> But, please think about making things simpler in general, cgroup
> is riddled with mis-designed complexities and memcg seems to be
> leading the charge at times.

Yes this is an evolution and it seems that we are slowly getting there.

>
> Thanks.

--
Michal Hocko
SUSE Labs