Subject: Re: [PATCH v5 00/14] kmem controller for memcg.
Posted by akpm on Thu, 18 Oct 2012 19:21:05 GMT
View Forum Message <> Reply to Message

On Thu, 18 Oct 2012 20:51:05 +0400
Glauber Costa <glommer@parallels.com> wrote:

> On 10/18/2012 02:11 AM, Andrew Morton wrote:
> > On Tue, 16 Oct 2012 14:16:37 +0400
> > Glauber Costa <glommer@parallels.com> wrote:
> >
> >> ...
> >>
> >> A general explanation of what this is all about follows:
> >>
> >> The kernel memory limitation mechanism for memcg concerns itself with
> >> disallowing potentially non-reclaimable allocations to happen in exaggerate
> >> quantities by a particular set of processes (cgroup). Those allocations could
> >> create pressure that affects the behavior of a different and unrelated set of
> >> processes.
> >>
> >> Its basic working mechanism is to annotate some allocations with the
> >> _GFP_KMEMCG flag. When this flag is set, the current process allocating will
> >> have its memcg identified and charged against. When reaching a specific limit,
> >> further allocations will be denied.
> >
> > The need to set _GFP_KMEMCG is rather unpleasing, and makes one wonder
> > "why didn't it just track all allocations".
> >
> This was raised as well by Peter Zijlstra during the memcg summit.

Firstly: please treat any question from a reviewer as an indication
that information was missing from the changelog or from code comments.
Ideally all such queries are addressed in later version of the patch
and changelog.

> The
> answer I gave to him still stands: There is a cost associated with it.
> We believe it comes down to a trade off situation. How much tracking a
> particular kind of allocation help vs how much does it cost.
>
> The free path is specially more expensive, since it will always incur in
> a page_cgroup lookup.

OK. But that is a quantitative argument, without any quantities! Do
we have even an estimate of what this cost will be? Perhaps it's the
case that, if well implemented, that cost will be acceptable. How do
we tell?

> > Does this mean that over time we can expect more sites to get the
> > _GFP_KMEMCG tagging?
>
> We have being doing kernel memory limitation for OpenVZ for a lot of
> times, using a quite different mechanism. What we do in this work (with
> slab included), allows us to achieve feature parity with that. It means
> it is good enough for production environments.

That's really good info.

> Whether or not more people will want other allocations to be tracked, I
> can't predict. What I do can say is that stack + slab is a very
> significant part of the memory one potentially cares about, and if
> anyone else ever have the need for more, it will come down to a
> trade-off calculation.

OK.

> > If so, are there any special implications, or do
> > we just go in, do the one-line patch and expect everything to work?
>
> With the infrastructure in place, it shouldn't be hard. But it's not
> necessarily a one-liner either. It depends on what are the pratical
> considerations for having that specific kind of allocation tied to a
> memcg. The slab, for instance, that follows this series, is far away
> from a one-liner: it is in fact, a 19-patch patch series.
>
>
>
> >
> > And how *accurate* is the proposed code? What percentage of kernel
> > memory allocations are unaccounted, typical case and worst case?
>
> With both patchsets applied, all memory used for the stack and most of
> the memory used for slab objects allocated in userspace process contexts
> are accounted.
>
> I honestly don't know which percentage of the total kernel memory this
> represents.

It sounds like the coverage will be good. What's left over? Random
get_free_pages() calls and interrupt-time slab allocations?

I suppose that there are situations in which network rx could consume
significant amounts of unaccounted memory?

> The accuracy for stack pages is very high: In this series, we don't move

> stack pages around when moving a task to other cgroups (for stack, it
> could be done), but other than that, all processes that pops up in a
> cgroup and stay there will have its memory accurately accounted.
>
> The slab is more complicated, and depends on the workload. It will be
> more accurate in workloads in which the level of object-sharing among
> cgroups is low. A container, for instance, is the perfect example of
> where this happen.
>
> >
> > All sorts of questions come to mind over this decision, but it was
> > unexplained. It should be, please. A lot!
> >
> >>
> >> ...
> >>
> >> Limits lower than
> >> the user limit effectively means there is a separate kernel memory limit that
> >> may be reached independently than the user limit. Values equal or greater than
> >> the user limit implies only that kernel memory is tracked. This provides a
> >> unified vision of "maximum memory", be it kernel or user memory.
> >>
> >
> > I'm struggling to understand that text much at all. Reading the
> > Documentation/cgroups/memory.txt patch helped.
> >
>
> Great. If you have any specific suggestions I can change that. Maybe I
> should just paste the documentation bit in here...

That's not a bad idea.