Subject: Re: [RFC 2/4] memcg: make it suck faster
Posted by Glauber Costa on Wed, 26 Sep 2012 08:53:21 GMT
View Forum Message <> Reply to Message

On 09/26/2012 01:02 AM, Andrew Morton wrote:
>> nomemcg : memcg compile disabled.
>> > base : memcg enabled, patch not applied.
>> > bypassed : memcg enabled, with patch applied.
>> >
>> > base bypassed
>> > User 109.12 105.64
>> > System 1646.84 1597.98
>> > Elapsed 229.56 215.76
>> >
>> > nomemcg bypassed
>> > User 104.35 105.64
>> > System 1578.19 1597.98
>> > Elapsed 212.33 215.76
>> >
>> > So as one can see, the difference between base and nomemcg in terms
>> > of both system time and elapsed time is quite drastic, and consistent
>> > with the figures shown by Mel Gorman in the Kernel summit. This is a
>> > ~ 7 % drop in performance, just by having memcg enabled. memcg functions
>> > appear heavily in the profiles, even if all tasks lives in the root
>> > memcg.
>> >
>> > With bypassed kernel, we drop this down to 1.5 %, which starts to fall
>> > in the acceptable range. More investigation is needed to see if we can
>> > claim that last percent back, but I believe at last part of it should
>> > be.
> Well that's encouraging. I wonder how many users will actually benefit
> from this - did I hear that major distros are now using memcg in some
> system-infrastructure-style code?
>

If they do, they actually be come "users of memcg". This here is aimed
at non-users of memcg, which given all the whining about it, it seems to
be plenty.

Also, I noticed, for instance, that libvirt is now creating memcg
hierarchies for lxc and qemu as placeholders, before you actually create
any vm or container.

There are two ways around that:
1) Have userspace in general to defer the creation of those directories
until they are actually going to use it, given the costs associated with
this.

2) defer our creation of memcg structures until the first task joins the
group, which then is the most clear signal possible that this is being
used.

> iirc, the idea of disabling memcg operations until someone enables a
> container had a couple of problems:
>
I'll need other people to jump in here and make it specific, but in general:

> a) certain boot-time initialisation issn't performed and
>

I am calling page_cgroup_init() at 1st memcg creation time.
The problem still exist that we will have tasks that are in LRUs but
with page_cgroup not filled. I handled this is this series by just not
testing this case, and assuming that empty page_cgroup == root_cgroup.

This can make bugs harder to find should they arise, but I'll argue that
it is worth it, given the gains.

> b) when memcg starts running for real, it expects that certain stats
>   gathering has been running since boot. If this is not the case,
>   those stats are wrong and stuff breaks.
>

I need specifics as well, but in general, my strategy lays in the
observation that all those statistics are already gathered in a global
level. We just report the global statistics when someone asks for it for
the root cgroup.

> It would be helpful if you could summarise these and similar issues
> and describe how they were addressed.
>

See above.
I would appreciate if anyone with a more specific and directed concern
would raise it.

>> >
>> > ...
>> >
>> > struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
>> > {
>> > + if (mem_cgroup_disabled())
>> > + return root_mem_cgroup;
> There would be some benefit in inlining the above instructions into the

> caller.
>
>> > return mem_cgroup_from_css(
>> >   cgroup_subsys_state(cont, mem_cgroup_subsys_id));
>> > }
> In fact the entire mem_cgroup_from_cont() could be inlined.
>

Indeed.

>> > struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p)
>> > {
>> > +
>> > + if (mem_cgroup_disabled())
>> > + return root_mem_cgroup;
> Ditto.
>

Indeetto.

>> > /*
>> >   * mm_update_next_owner() may clear mm->owner to NULL
>> >   * if it races with swapoff, page migration, etc.
>> >
>> > ...
>> >