Subject: Re: [RFC 2/4] memcg: make it suck faster
Posted by akpm on Tue, 25 Sep 2012 21:02:36 GMT
View Forum Message <> Reply to Message

On Tue, 25 Sep 2012 12:52:51 +0400
Glauber Costa <glommer@parallels.com> wrote:

> It is an accepted fact that memcg sucks. But can it suck faster? Or in
> a more fair statement, can it at least stop draining everyone's
> performance when it is not in use?
>
> This experimental and slightly crude patch demonstrates that we can do
> that by using static branches to patch it out until the first memcg
> comes to life. There are edges to be trimmed, and I appreciate comments
> for direction. In particular, the events in the root are not fired, but
> I believe this can be done without further problems by calling a
> specialized event check from mem_cgroup_newpage_charge().
>
> My goal was to have enough numbers to demonstrate the performance gain
> that can come from it. I tested it in a 24-way 2-socket Intel box, 24 Gb
> mem. I used Mel Gorman's pft test, that he used to demonstrate this
> problem back in the Kernel Summit. There are three kernels:
>
> 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?

iirc, the idea of disabling memcg operations until someone enables a container had a couple of problems:

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

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.

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

>
> ...
>
> 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.

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

Ditto.

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