Subject: Re: [PATCH v2 09/11] memcg: propagate kmem limiting information to children
Posted by Michal Hocko on Fri, 17 Aug 2012 09:35:04 GMT
View Forum Message <> Reply to Message

On Fri 17-08-12 13:15:47, Glauber Costa wrote:
> On 08/17/2012 01:00 PM, Michal Hocko wrote:
> > On Thu 09-08-12 17:01:17, Glauber Costa wrote:
> >> The current memcg slab cache management fails to present satisfatory
> >> hierarchical behavior in the following scenario:
> >>
> >> -> /cgroups/memory/A/B/C
> >>
> >> * kmem limit set at A,
> >> * A and B have no tasks,
> >> * span a new task in in C.
> >>
> >> Because kmem_accounted is a boolean that was not set for C, no
> >> accounting would be done. This is, however, not what we expect.
> >>
> >> The basic idea, is that when a cgroup is limited, we walk the tree
> >> upwards
> >
> > Isn't it rather downwards? We start at A and then mark all children so
> > we go down the tree. Moreover the walk is not atomic wrt. parallel
> > charges nor to a new child creation. First one seems to be acceptable
> > as the charges go to the root. The second one requires cgroup_lock.
> >
>
> Yes, it is downwards. I've already noticed that yesterday and updated
> in my tree.
>
> As for the lock, can't we take set_limit lock in cgroup creation just
> around the place that updates that field in the child? It is a lot more
> fine grained - everything except the dead bkl is - and what we're
> actually protecting is the limit.

That should work as well. It is less obvious because we are not
considering the parent limit (maybe we should rename the lock but that
is just a detail).

> If you prefer, I can use cgroup lock just fine. But then I won't sleep
> at night and probably pee my pants, which is something I don't do for at
> least two decades now.

Heh, please no, I would feel terrible then

> > It also seems that you are missing memcg_kmem_account_parent in

> > mem_cgroup_create (use_hierarchy path) if memcg_kmem_is_accounted(parent).
> >
>
> You mean when we create a cgroup ontop of an already limited parent?

I would prefer bellow but yes
A (a) - B (a, pa)
      - C (new)

> Humm, you are very right.
>
> > Some further "wording" comments below. Other than that the patch looks
> > correct.
> >
> >> (something Kame and I already thought about doing for other
> >> purposes), and make sure that we store the information about the parent
> >> being limited in kmem_accounted (that is turned into a bitmap: two
> >> booleans would not be space efficient).
> >
> > Two booleans even don't serve the purpose because you want to test this
> > atomically, right?
> >
>
> Well, yes, we have that extra problem as well.
> >> The code for that is taken from sched/core.c. My reasons for not
> >> putting it into a common place is to dodge the type issues that would
> >> arise from a common implementation between memcg and the scheduler -
> >> but I think that it should ultimately happen, so if you want me to do
> >> it now, let me know.
> >
> > Is this really relevant for the patch?
> >
>
> Not at all. Besides not being relevant, it is also not true, since I now
> use the memcg iterator. I would prefer the tree walk instead of having
> to cope with the order imposed by the memcg iterator, but we add
> less code this way...
>
> Again, already modified that in my yesterday's update.

OK

> >> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> >> index 3216292..3d30b79 100644
> >> --- a/mm/memcontrol.c
> >> +++ b/mm/memcontrol.c
> >> @@ -295,7 +295,8 @@ struct mem_cgroup {
> >>   * Should the accounting and control be hierarchical, per subtree?

> >>   */
> >>   bool use_hierarchy;
> >> - bool kmem_accounted;
> >> +
> >> + unsigned long kmem_accounted; /* See KMEM_ACCOUNTED_*, below */
> >>
> >>   bool oom_lock;
> >>   atomic_t under_oom;
> >> @@ -348,6 +349,38 @@ struct mem_cgroup {
> >> #endif
> >> };
> >>
> >> +enum {
> >> + KMEM_ACCOUNTED_THIS, /* accounted by this cgroup itself */
> >> + KMEM_ACCOUNTED_PARENT, /* accounted by any of its parents. */
> > >
> > > How it can be accounted by its parent, the charge doesn't go downwards.
> > > Shouldn't it rather be /* a parent is accounted */
> > >
> > indeed.
> >
> >> +};
> >> +
> >> +#ifdef CONFIG_MEMCG_KMEM
> >> +static bool memcg_kmem_account(struct mem_cgroup *memcg)
> > >
> > > memcg_kmem_set_account? It matches _clear_ counterpart and it makes
> > > obvious that the value is changed actually.
> > >
> >
> > Ok.
> >
> > > [...]
> >> +static bool memcg_kmem_is_accounted(struct mem_cgroup *memcg)
> >> +{
> >> + return test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted);
> >> +}
> >> +
> >> +static void memcg_kmem_account_parent(struct mem_cgroup *memcg)
> > >
> > > same here _set_parent
> > >
> >
> > Ok, agreed.

Thanks

>

> > [...]
> >> @@ -614,7 +647,7 @@ EXPORT_SYMBOL(__memcg_kmem_free_page);
> >>
> >> static void disarm_kmem_keys(struct mem_cgroup *memcg)
> >> {
> >> - if (memcg->kmem_accounted)
> >> + if (test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted))
> >
> > memcg_kmem_is_accounted. I do not see any reason to open code this.
> >
>
> ok.
>
> >> #ifdef CONFIG_MEMCG_KMEM
> >> - /*
> >> - * Once enabled, can't be disabled. We could in theory disable it if we
> >> - * haven't yet created any caches, or if we can shrink them all to
> >> - * death. But it is not worth the trouble.
> >> - */
> >> + struct mem_cgroup *iter;
> >> +
> >>   mutex_lock(&set_limit_mutex);
> >> - if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
> >> + if ((val != RESOURCE_MAX) && memcg_kmem_account(memcg)) {
> >> +
> >> + /*
> >> +   * Once enabled, can't be disabled. We could in theory disable
> >> +   * it if we haven't yet created any caches, or if we can shrink
> >> +   * them all to death. But it is not worth the trouble
> >> +   */
> >>   static_key_slow_inc(&memcg_kmem_enabled_key);
> >> - memcg->kmem_accounted = true;
> >> +
> >> + if (!memcg->use_hierarchy)
> >> +   goto out;
> >> +
> >> + for_each_mem_cgroup_tree(iter, memcg) {
> >
> > for_each_mem_cgroup_tree does respect use_hierarchy so the above
> > shortcut is not necessary. Dunno but IMHO we should get rid of explicit
> > tests as much as possible. This doesn't look like a hot path anyway.
> >
>
> I can't remember any reason for doing so other than gaining some time.
> I will remove it.

Well it involves a bit more code because you would basically do expand
to a loop which does one iteration (continue) and terminates also take

and drop the reference on the group. That all seems unnecessary but as I
said this is not a hot path and we better get rid of direct checks.
I am not insisting on this so use your good taste...

>
> >> +   if (iter == memcg)
> >> +   continue;
> >> +   memcg_kmem_account_parent(iter);
> >> + }
> >> + } else if ((val == RESOURCE_MAX) && memcg_kmem_clear_account(memcg)) {
> >
> > Above you said "Once enabled, can't be disabled." and now you can
> > disable it? Say you are a leaf group with non accounted parents. This
> > will clear the flag and so no further accounting is done. Shouldn't
> > unlimited mean that we will never reach the limit? Or am I missing
> > something?
> >
>
> You are missing something, and maybe I should be more clear about that.
> The static branches can't be disabled (it is only safe to disable them
> from disarm_static_branches(), when all references are gone). Note that
> when unlimited, we flip bits, do a transversal, but there is no mention
> to the static branch.

My little brain still doesn't get this. I wasn't concerned about static
branches. I was worried about memcg_can_account_kmem which will return
false now, doesn't it.

>
> The limiting can come and go at will.
>
> >> +
> >> + if (!memcg->use_hierarchy)
> >> +   goto out;
> >> +
> >> + for_each_mem_cgroup_tree(iter, memcg) {
> >> +   struct mem_cgroup *parent;
> >> +
> >> +   if (iter == memcg)
> >> +   continue;
> >> +   /*
> >> +   * We should only have our parent bit cleared if none
> >> +   * of our parents are accounted. The transversal order
> >> +   * of our iter function forces us to always look at the
> >> +   * parents.
> >> +   */
> >> +   parent = parent_mem_cgroup(iter);
> >> +   for (; parent != memcg; parent = parent_mem_cgroup(iter))

> >> +   if (memcg_kmem_is_accounted(parent))
> >> +   goto noclear;
> >> +   memcg_kmem_clear_account_parent(iter);
> >
> > Brain hurts...
> > Yes we are iterating in the creation ordering so we cannot rely on the
> > first encountered accounted memcg
> > A(a) - B - D
> >   - C (a) - E
>
> That's why I said I preferred the iterator the scheduler uses. The
> actual transverse code was much simpler, because it will stop at an
> unlimited parent. But this is the only drawback I see in the memcg
> iterator, so I decided that just documenting this "interesting" piece of
> code well would do...

I was just complaining that more specific comment would be much more
helpful... The ordering might be non-trivial for those who are not
familiar with cgroup internals because id doesn't tell you much.

--
Michal Hocko
SUSE Labs
