Subject: Re: [PATCH v4 04/14] kmem accounting basic infrastructure
Posted by Glauber Costa on Fri, 12 Oct 2012 07:36:38 GMT
View Forum Message <> Reply to Message

On 10/11/2012 02:11 PM, Michal Hocko wrote:
> On Mon 08-10-12 14:06:10, Glauber Costa wrote:
>> This patch adds the basic infrastructure for the accounting of the slab
>> caches. To control that, the following files are created:
>>
>> * memory.kmem.usage_in_bytes
>> * memory.kmem.limit_in_bytes
>> * memory.kmem.failcnt
>> * memory.kmem.max_usage_in_bytes
>>
>> They have the same meaning of their user memory counterparts. They
>> reflect the state of the "kmem" res_counter.
>>
>> Per cgroup slab memory accounting is not enabled until a limit is set
>
> s/slab/kmem/ right?
>
right.

>> +static int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
>> +{
>> + int ret = -EINVAL;
>> +#ifdef CONFIG_MEMCG_KMEM
>> + struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
>> + /*
>> + * For simplicity, we won't allow this to be disabled. It also can't
>> + * be changed if the cgroup has children already, or if tasks had
>> + * already joined.
>> + *
>> + * If tasks join before we set the limit, a person looking at
>> + * kmem.usage_in_bytes will have no way to determine when it took
>> + * place, which makes the value quite meaningless.
>> + *
>> + * After it first became limited, changes in the value of the limit are
>> + * of course permitted.
>> + *
>> + * Taking the cgroup_lock is really offensive, but it is so far the only
>> + * way to guarantee that no children will appear. There are plenty of
>> + * other offenders, and they should all go away. Fine grained locking
>> + * is probably the way to go here. When we are fully hierarchical, we
>> + * can also get rid of the use_hierarchy check.
>> + */
>> + cgroup_lock();
>> + mutex_lock(&set_limit_mutex);

>> + if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
>
> Just a nit but wouldn't memcg_kmem_is_accounted(memcg) be better than
> directly checking kmem_accounted?
> Besides that I am not sure I fully understand RESOURCE_MAX test. Say I
> want to have kmem accounting for monitoring so I do
> echo -1 > memory.kmem.limit_in_bytes
>
> so you set the value but do not activate it. Isn't this just a reminder
> from the time when the accounting could be deactivated?
>

No, not at all.

I see you have talked about that in other e-mails, (I was on sick leave
yesterday), so let me consolidate it all here:

What we discussed before, regarding to echo -1 > ... was around the
disable code, something that we no longer allow. So now, if you will
echo -1 to that file *after* it is limited, you get in track only mode.

But for you to start that, you absolutely have to write something
different than -1.

Just one example: libcgroup, regardless of how lame we think it is in
this regard, will write to all cgroup files by default when a file is
updated. If you haven't written anything, it will still write the same
value that the file had before.

This means that an already deployed libcg-managed installation will
suddenly enable kmem for every cgroup. Sure this can be fixed in
userspace, but:

1) There is no reason to break it, if we can
2) It is perfectly reasonable to expect that if you write to a file the
same value that was already there, nothing happens.

I'll update the docs to say that you can just write -1 *after* it is
limited, but i believe enabling it has to be a very clear transition,
for sanity's sake.

>> + if (cgroup_task_count(cont) || (memcg->use_hierarchy &&
>> +   !list_empty(&cont->children))) {
>> +   ret = -EBUSY;
>> +   goto out;
>> + }
>> + ret = res_counter_set_limit(&memcg->kmem, val);
>

> VM_BUG_IN(ret) ?
> There shouldn't be any usage when you enable it or something bad is
> going on.
>
Good point, this is indeed an impossible scenario I was just being overcautious about.