Subject: Re: [PATCH v4 23/25] memcg: propagate kmem limiting information to children
Posted by Glauber Costa on Tue, 19 Jun 2012 08:54:35 GMT
View Forum Message <> Reply to Message

```
On 06/19/2012 12:35 PM, Glauber Costa wrote:
> On 06/19/2012 04:16 AM, Kamezawa Hiroyuki wrote:
>> (2012/06/18 21:43), Glauber Costa wrote:
>>> On 06/18/2012 04:37 PM, Kamezawa Hiroyuki wrote:
>>>> (2012/06/18 19:28), 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 empty taskwise
>>>>> * bash in C does find /
>>>>>
>>>>> Because kmem_accounted is a boolean that was not set for C, no accounting
>>>>> would be done. This is, however, not what we expect.
>>>>>
>>>>
>>>> Hmm....do we need this new routines even while we have mem_cgroup_iter() ?
>>>>
>>>> Doesn't this work ?
>>>>
>>>>  struct mem_cgroup {
>>>>   .....
>>>>   bool kmem_accounted_this;
>>>>   atomic_t kmem_accounted;
>>>>   ....
>>>>  }
>>>>
>>>> at set limit
>>>>
>>>>  ....set_limit(memcg) {
>>>>
>>>>   if (newly accounted) {
>>>>    mem_cgroup_iter() {
>>>>     atomic_inc(&iter->kmem_accounted)
>>>>    }
>>>>   } else {
>>>>    mem_cgroup_iter() {
>>>>     atomic_dec(&iter->kmem_accounted);
>>>>    }
>>>>  }
>>>>
```

>>>>
>>>> hm ? Then, you can see kmem is accounted or not by
atomic_read(&memcg->kmem_accounted);
>>>>
>>>
>>> Accounted by itself / parent is still useful, and I see no reason to use
>>> an atomic + bool if we can use a pair of bits.
>>>
>>> As for the routine, I guess mem_cgroup_iter will work... It does a lot
>>> more than I need, but for the sake of using what's already in there, I
>>> can switch to it with no problems.
>>>
>>
>> Hmm. please start from reusing existing routines.
>> If it's not enough, some enhancement for generic cgroup will be welcomed
>> rather than completely new one only for memcg.
>>
>
> And now that I am trying to adapt the code to the new function, I
> remember clearly why I done this way. Sorry for my failed memory.
>
> That has to do with the order of the walk. I need to enforce hierarchy,
> which means whenever a cgroup has !use_hierarchy, I need to cut out that
> branch, but continue scanning the tree for other branches.
>
> That is a lot easier to do with depth-search tree walks like the one
> proposed in this patch. for_each_mem_cgroup() seems to walk the tree in
> css-creation order. Which means we need to keep track of parents that
> has hierarchy disabled at all times (can be many), and always test for
> ancestorship - which is expensive, but I don't particularly care.
>
> But I'll give another shot with this one.
>

Humm, silly me. I was believing the hierarchical settings to be more
flexible than they really are.

I thought that it could be possible for a children of a parent with
use_hierarchy = 1 to have use_hierarchy = 0.

It seems not to be the case. This makes my life a lot easier.