Subject: Re: [RFC 5/7] use percpu_counters for res_counter usage
Posted by Glauber Costa on Fri, 30 Mar 2012 12:59:32 GMT
View Forum Message <> Reply to Message

> diff --git a/kernel/res_counter.c b/kernel/res_counter.c
>> index 052efaf..8a99943 100644
>> --- a/kernel/res_counter.c
>> +++ b/kernel/res_counter.c
>> @@ -28,9 +28,28 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
>> int ret = 0;
>> u64 usage;
>>
>> + rcu_read_lock();
>> +
>
>
> Hmm... isn't it better to synchronize percpu usage to the main counter
> by smp_call_function() or some at set limit ? after set 'global' mode ?

Yes. I think it should be done after global mode is set.
My idea is to flush all the percpu data, and then start treating that
essentially as a res_counter is today.

>
> set global mode
> smp_call_function(drain all pcp counters to main counter)
> set limit.
> unset global mode
>
>> + if (val< 0) {
>> + percpu_counter_add(&c->usage_pcp, val);
>> + rcu_read_unlock();
>> + return 0;
>> + }
>
>
> Memo:
> memcg's uncharge path is batched ....so..it will be bigger than
> percpu_counter_batch() in most of cases. (And lock conflict is enough low.)
>

I don't get what you mean.
It is batched, because charge is batched. If we un-batch one, we
un-batch another. And if we don't we don't do for any.

>> +
>> + usage = percpu_counter_read(&c->usage_pcp);
>> +

>> + if (percpu_counter_read(&c->usage_pcp) + val<
>> +   (c->limit + num_online_cpus() * percpu_counter_batch)) {
>
>
> c->limit - num_online_cpus() * percpu_counter_batch ?
>
> Anyway, you can pre-calculate this value at cpu hotplug event..

Beautiful. Haven't thought about that.

Thanks.

>
>> + percpu_counter_add(&c->usage_pcp, val);
>> + rcu_read_unlock();
>> + return 0;
>> + }
>> +
>> + rcu_read_unlock();
>> +
>>   raw_spin_lock(&c->usage_pcp.lock);
>>
>> - usage = c->usage;
>> + usage = __percpu_counter_sum_locked(&c->usage_pcp);
>
>
> Hmm.... this part doesn't seem very good.
> I don't think for_each_online_cpu() here will not be a way to the final win.
> Under multiple hierarchy, you may need to call for_each_online_cpu() in each level.
>
> Can't you update percpu counter's core logic to avoid using for_each_online_cpu() ?
> For example, if you know what cpus have caches, you can use that cpu mask...

A mask should work, yes.

Flipping a bit when a cpu update its data shouldn't hurt that much.
There is cache sharing and everything, but in most cases we won't be
really making it dirty.

> Memo:
> Current implementation of memcg's percpu counting is reserving usage before its real use.
> In usual, the kernel don't have to scan percpu caches and just drain caches from cpus
> reserving usages if we need to cancel reserved usages. (And it's automatically canceled
> when cpu's memcg changes.)
>
> And 'reserving' avoids caching in multi-level counters,....it updates multiple counters
> in batch and memcg core don't need to walk res_counter ancestors in fast path.
>

> Considering res_counter's characteristics
> - it has _hard_ limit
> - it can be tree and usages are propagated to ancestors
> - all ancestors has hard limit.
>
> Isn't it better to generalize 'reserving resource' model ?

It would be nice to see an implementation of that as well to see how it
will turn up.

Meanwhile, points to consider over the things you raised:

1) I think if we use something like the global flag as I described, we
can pretty much guarantee hard limits in the memcg code.

2) Specially because it is a tree with usage propagated to the
ancestors, is that I went with a percpu approach.

See, We can reserve as many pages as we want. This only happens in the
level we are reserving.
If two different cgroups that share an ancestor reserve at the same
time, in different cpus, we would expect to see a more parallel
behavior. Instead, we'll have contention in the ancestor.

It gets even worse if the ancestor is not under pressure, because there
is no reason for the contention. You will cache the leaves, but that
won't help with the intermediate levels.

However, there are some points I admit:

The pressure-behavior with a pure per-cpu proposal is worse, because
then when you are under pressure, you are billing page-by-page, instead
of in bulks.

So maybe what we need is to make the res_counter batched by default - or
provided a res_counter_charge_batched() and
convert memcg for that. Then we can have a cache per res_counter. Two
children of the same res_counter will then both be able to consume their
parent stock, and we can maybe move forward without contention in this case.

> You can provide 'precise usage' to the user by some logic.
>
>>
>> if (usage + val> c->limit) {
>> c->failcnt++;
>> @@ -39,9 +58,9 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
>> goto out;
>> }

>>
>
>
> Can we user synchronize_rcu() under spin_lock ?
> I don't think this synchronize_rcu() is required.
> percpu counter is not precise in its nature. __percpu_counter_sum_locked() will be enough.

I am starting to think it is not needed as well, specially here.
My goal was to make sure we don't have other per-cpu updaters
when we are trying to grab the final value. But guess that the only
place in which it really
matters is when we do limit testing.
