Subject: Re: [PATCH v2 5/5] decrement static keys on real destroy time
Posted by KAMEZAWA Hiroyuki on Wed, 25 Apr 2012 00:22:37 GMT
View Forum Message <> Reply to Message

(2012/04/24 20:41), Glauber Costa wrote:

```
> On 04/23/2012 11:40 PM, KAMEZAWA Hiroyuki wrote:
>> (2012/04/24 4:37), Glauber Costa wrote:
>>
>>> We call the destroy function when a cgroup starts to be removed,
>>> such as by a rmdir event.
>>>
>>> However, because of our reference counters, some objects are still
>>> inflight. Right now, we are decrementing the static_keys at destroy()
>>> time, meaning that if we get rid of the last static_key reference,
>>> some objects will still have charges, but the code to properly
>>> uncharge them won't be run.
>>>
>>> This becomes a problem specially if it is ever enabled again, because
>>> now new charges will be added to the staled charges making keeping
>>> it pretty much impossible.
>>>
>>> We just need to be careful with the static branch activation:
>>> since there is no particular preferred order of their activation,
>>> we need to make sure that we only start using it after all
>>> call sites are active. This is achieved by having a per-memcg
>>> flag that is only updated after static_key_slow_inc() returns.
>>> At this time, we are sure all sites are active.
>>>
>>> This is made per-memcg, not global, for a reason:
>>> it also has the effect of making socket accounting more
>>> consistent. The first memcg to be limited will trigger static_key()
>>> activation, therefore, accounting. But all the others will then be
>>> accounted no matter what. After this patch, only limited memcgs
>>> will have its sockets accounted.
>>>
>>> [v2: changed a tcp limited flag for a generic proto limited flag ]
>>> [v3: update the current active flag only after the static_key update ]
>>>
>>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>>
>>
>> Acked-by: KAMEZAWA Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
>>
>> A small request below.
>>
>> <snip>
>>
```

```
>>
>>> +   * ->activated needs to be written after the static_key update.
>>> +   *  This is what guarantees that the socket activation function
>>> +   *  is the last one to run. See sock_update_memcg() for details,
>>> +   *  and note that we don't mark any socket as belonging to this
>>> +   *  memcg until that flag is up.
>>> +   *
>>> +   *  We need to do this, because static_keys will span multiple
>>> +   *  sites, but we can't control their order. If we mark a socket
>>> +   *  as accounted, but the accounting functions are not patched in
>>> +   *  yet, we'll lose accounting.
>>> +   *
>>> +   *  We never race with the readers in sock_update_memcg(), because
>>> +   *  when this value change, the code to process it is not patched in
>>> +   *  yet.
>>> +   */
>>> +  mutex_lock(&tcp_set_limit_mutex);
>>
>>
>> Could you explain for what this mutex is in above comment ?
>>
> This is explained at the site where the mutex is defined.
> If you still want me to mention it here, or maybe expand the explanation
> there, I surely can.
>
```

Ah, I think it's better to mention one more complicated race.
Let me explain.

Assume we don't have tcp_set_limit_mutex. And jump_label is not activated yet
i.e. memcg_socket_limit_enabled->count == 0.

When a user updates limit of 2 cgroups at once, following happens.

```
 CPU A    CPU B

    if (cg_proto->activated)    if (cg->proto_activated)
        static_key_inc()                static_key_inc()
        => set counter 0->1             => set counter 1->2, return immediately.
           => hold mutex                => cg_proto->activated = true.
              => overwrite jmps.
```

Then, without mutex, activated/active may be set 'true' before the end
of jump_label modification.

Thanks,
-Kame