Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by akpm on Wed, 16 May 2012 21:06:37 GMT
View Forum Message <> Reply to Message

On Fri, 11 May 2012 17:11:17 -0300
Glauber Costa <glommer@parallels.com> 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.
>
> ...
>
> @@ -107,10 +104,31 @@ static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
> tcp->tcp_prot_mem[i] = min_t(long, val >> PAGE_SHIFT,
> net->ipv4.sysctl_tcp_mem[i]);
>
> - if (val == RESOURCE_MAX && old_lim != RESOURCE_MAX)
> - static_key_slow_dec(&memcg_socket_limit_enabled);
> - else if (old_lim == RESOURCE_MAX && val != RESOURCE_MAX)
> - static_key_slow_inc(&memcg_socket_limit_enabled);
> + if (val == RESOURCE_MAX)
> + cg_proto->active = false;
> + else if (val != RESOURCE_MAX) {
> + /*

```
> +   * ->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.
> +   */
> +  if (!cg_proto->activated) {
> +   static_key_slow_inc(&memcg_socket_limit_enabled);
> +   cg_proto->activated = true;
> +  }
```

If two threads run this code concurrently, they can both see
cg_proto->activated==false and they will both run
static_key_slow_inc().

Hopefully there's some locking somewhere which prevents this, but it is
unobvious. We should comment this, probably at the cg_proto.activated
definition site. Or we should fix the bug ;)

```
> +  cg_proto->active = true;
> + }
>
>   return 0;
>  }
>
> ...
>
```
