Subject: Re: [PATCH v3 2/2] decrement static keys on real destroy time
Posted by Tejun Heo on Thu, 26 Apr 2012 21:39:16 GMT
View Forum Message <> Reply to Message

Hello, Glauber.

Overall, I like this approach much better. Just some nits below.

On Thu, Apr 26, 2012 at 06:24:23PM -0300, Glauber Costa wrote:
> @@ -4836,6 +4851,18 @@ static void free_work(struct work_struct *work)
> int size = sizeof(struct mem_cgroup);
>
> memcg = container_of(work, struct mem_cgroup, work_freeing);
> + /*
> + * We need to make sure that (at least for now), the jump label
> + * destruction code runs outside of the cgroup lock. It is in theory
> + * possible to call the cgroup destruction function outside of that
> + * lock, but it is not yet done. rate limiting plus the deferred
> + * interface for static_branch destruction guarantees that it will
> + * run through schedule_work(), therefore, not holding any cgroup
> + * related lock (this is, of course, until someone decides to write
> + * a schedule_work cgroup :p)
> + */

Isn't the above a bit too verbose? Wouldn't just stating the locking
dependency be enough?

> + disarm_static_keys(memcg);
> if (size < PAGE_SIZE)
> kfree(memcg);
> else
> diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
> index 1517037..7790008 100644
> --- a/net/ipv4/tcp_memcontrol.c
> +++ b/net/ipv4/tcp_memcontrol.c
> @@ -54,6 +54,8 @@ int tcp_init_cgroup(struct mem_cgroup *memcg, struct cgroup_subsys
*ss)
> cg_proto->sysctl_mem = tcp->tcp_prot_mem;
> cg_proto->memory_allocated = &tcp->tcp_memory_allocated;
> cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;
> + cg_proto->active = false;
> + cg_proto->activated = false;

Isn't the memory zallocd? I find 0 / NULL / false inits unnecessary
and even misleading (can the memory be non-zero here?). Another side
effect is that it tends to get out of sync as more fields are added.

> +/*

> + * This is to prevent two writes arriving at the same time
> + * at kmem.tcp.limit_in_bytes.
> + *
> + * There is a race at the first time we write to this file:
> + *
> + * - cg_proto->activated == false for all writers.
> + * - They all do a static_key_slow_inc().
> + * - When we are finally read to decrement the static_keys,
                    ^
                    ready

> + *   we'll do it only once per activated cgroup. So we won't
> + *   be able to disable it.
> + *
> + *   Also, after the first caller increments the static_branch
> + *   counter, all others will return right away. That does not mean,
> + *   however, that the update is finished.
> + *
> + *   Without this mutex, it would then be possible for a second writer
> + *   to get to the update site, return

I kinda don't follow the above sentence.

> + *   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.

Isn't this something which should be solved from static_keys API? Why is this being worked around from memcg? Also, I again hope that the explanation is slightly more concise.

Thanks.

--
tejun