Subject: [PATCH v3 0/2] fix problem with static_branch() for sock memcg
Posted by Glauber Costa on Thu, 26 Apr 2012 21:24:21 GMT
View Forum Message <> Reply to Message

Hi,

While trying to fulfill's Christoph's request for using static_branches
to do part of the role of number_of_cpusets in the cpuset cgroup, I took
a much more extensive look at the cpuset code (Thanks Christoph).

I started to feel that removing the cgroup_lock() from cpuset's
destroy is not as safe as I first imagined. At the very best, is not safe
enough to be bundled in a bugfix and deserves its own analysis.

I started then to consider another approach. While I voiced many times
that I would not like to do deferred updates for the static_branches, doing
that during destroy time would be perfectly acceptable IMHO (creation is
another story). In a summary, we are effectively calling the static_branch
updates only when the last reference to the memcg is gone. And that is
already asynchronous by nature, and we cope well with that.

In memcg, it turns out that we already do deferred freeing of the memcg
structure depending on the size of struct mem_cgroup.

My proposal is to always do that, and then we get a worker more or less
for free. Patch 2 is basically the same I had posted before, with minor
adaptations, plus the addition of a commentary explaining a race as
requested by Kame.

Let me know if this is acceptable.

Thanks

Glauber Costa (2):
  Always free struct memcg through schedule_work()
  decrement static keys on real destroy time

 include/net/sock.h | 9 ++++++
 mm/memcontrol.c | 54 ++++++++++++++++++++++++++++++--------
 net/ipv4/tcp_memcontrol.c | 70 ++-
 3 files changed, 113 insertions(+), 20 deletions(-)

--
1.7.7.6

Subject: [PATCH v3 1/2] Always free struct memcg through schedule_work()

Right now we free struct memcg with kfree right after a
rcu grace period, but defer it if we need to use vfree() to get
rid of that memory area. We do that by need, because we need vfree
to be called in a process context.

This patch unifies this behavior, by ensuring that even kfree will
happen in a separate thread. The goal is to have a stable place to
call the upcoming jump label destruction function outside the realm
of the complicated and quite far-reaching cgroup lock (that can't be
held when calling neither the cpu_hotplug.lock nor the jump_label_mutex)

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Tejun Heo <tj@kernel.org>
CC: Li Zefan <lizefan@huawei.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Michal Hocko <mhocko@suse.cz>
---
 mm/memcontrol.c | 24 ++++++++++++++++++-----------
 1 files changed, 13 insertions(+), 11 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 7832b4d..b0076cc 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -245,8 +245,8 @@ struct mem_cgroup {
    */
   struct rcu_head rcu_freeing;
   /*
-   * But when using vfree(), that cannot be done at
-   * interrupt time, so we must then queue the work.
+   * We also need some space for a worker in deferred freeing.
+   * By the time we call it, rcu_freeing is not longer in use.
    */
   struct work_struct work_freeing;
 };
@@ -4826,23 +4826,28 @@ out_free:
 }

 /*
- * Helpers for freeing a vzalloc()ed mem_cgroup by RCU,
+ * Helpers for freeing a kmalloc()ed/vzalloc()ed mem_cgroup by RCU,
  * but in process context. The work_freeing structure is overlaid
  * on the rcu_freeing structure, which itself is overlaid on memsw.
  */
-static void vfree_work(struct work_struct *work)

```
+static void free_work(struct work_struct *work)
 {
  struct mem_cgroup *memcg;
+ int size = sizeof(struct mem_cgroup);

  memcg = container_of(work, struct mem_cgroup, work_freeing);
- vfree(memcg);
+ if (size < PAGE_SIZE)
+  kfree(memcg);
+ else
+  vfree(memcg);
 }
-static void vfree_rcu(struct rcu_head *rcu_head)
+
+static void free_rcu(struct rcu_head *rcu_head)
 {
  struct mem_cgroup *memcg;

  memcg = container_of(rcu_head, struct mem_cgroup, rcu_freeing);
- INIT_WORK(&memcg->work_freeing, vfree_work);
+ INIT_WORK(&memcg->work_freeing, free_work);
  schedule_work(&memcg->work_freeing);
 }

@@ -4868,10 +4873,7 @@ static void __mem_cgroup_free(struct mem_cgroup *memcg)
   free_mem_cgroup_per_zone_info(memcg, node);

  free_percpu(memcg->stat);
- if (sizeof(struct mem_cgroup) < PAGE_SIZE)
-  kfree_rcu(memcg, rcu_freeing);
- else
-  call_rcu(&memcg->rcu_freeing, vfree_rcu);
+ call_rcu(&memcg->rcu_freeing, free_rcu);
 }

 static void mem_cgroup_get(struct mem_cgroup *memcg)
--
1.7.7.6
```