
Subject: [PATCH v4 05/25] memcg: Always free struct memcg through
schedule_work()

Posted by [Glauber Costa](#) on Mon, 18 Jun 2012 10:27:58 GMT

[View Forum Message](#) <> [Reply to Message](#)

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 file changed, 13 insertions(+), 11 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index e3b528e..ce15be4 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)
--
1.7.10.2

```
