Subject: [PATCH] cgroup: Convert synchronize_rcu to call_rcu in cgroup_attach_task
Posted by Colin Cross on Mon, 22 Nov 2010 04:06:07 GMT
View Forum Message <> Reply to Message

The synchronize_rcu call in cgroup_attach_task can be very expensive. All fastpath accesses to task->cgroups that expect task->cgroups not to change already use task_lock() or cgroup_lock() to protect against updates, and, in cgroup.c, only the CGROUP_DEBUG files have RCU read-side critical sections.

sched.c uses RCU read-side-critical sections on task->cgroups, but only to ensure that a dereference of task->cgroups does not become invalid, not that it doesn't change.

This patch adds a function put_css_set_rcu, which delays the put until after a grace period has elapsed. This ensures that any RCU read-side critical sections that dereferenced task->cgroups in sched.c have completed before the css_set is deleted. The synchronize_rcu()/put_css_set() combo in cgroup_attach_task() can then be replaced with put_css_set_rcu().

Also converts the CGROUP_DEBUG files that access current->cgroups to use task_lock(current) instead of rcu_read_lock().

Signed-off-by: Colin Cross <ccross@android.com>

This version fixes the problems with the previous patch by keeping the use of RCU in cgroup_attach_task, but allowing cgroup_attach_task to return immediately by deferring the final put_css_reg to an rcu callback.

```
 include/linux/cgroup.h |    4 +++
 kernel/cgroup.c        |   58 ++++++++++++++++++++++++++++++++++++++----------
 2 files changed, 50 insertions(+), 12 deletions(-)

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index ed4ba11..fd26218 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -287,6 +287,10 @@ struct css_set {

   /* For RCU-protected deletion */
```

```
  struct rcu_head rcu_head;
+
+ /* For RCU-delayed puts */
+ atomic_t delayed_put_count;
+ struct work_struct delayed_put_work;
 };

 /*
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 66a416b..c7348e7 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -298,7 +298,8 @@ static int cgroup_init_idr(struct cgroup_subsys *ss,

 /* css_set_lock protects the list of css_set objects, and the
  * chain of tasks off each css_set.  Nests outside task->alloc_lock
- * due to cgroup_iter_start() */
+ * due to cgroup_iter_start().  Never locked in irq context, so
+ * the non-irq variants of write_lock and read_lock are used. */
 static DEFINE_RWLOCK(css_set_lock);
 static int css_set_count;

@@ -396,6 +397,39 @@ static inline void put_css_set_taskexit(struct css_set *cg)
  __put_css_set(cg, 1);
 }

+/* work function, executes in process context */
+static void __put_css_set_rcu(struct work_struct *work)
+{
+ struct css_set *cg;
+ cg = container_of(work, struct css_set, delayed_put_work);
+
+ while (atomic_add_unless(&cg->delayed_put_count, -1, 0))
+  put_css_set(cg);
+}
+
+/* rcu callback, executes in softirq context */
+static void _put_css_set_rcu(struct rcu_head *obj)
+{
+ struct css_set *cg = container_of(obj, struct css_set, rcu_head);
+
+ /* the rcu callback happens in softirq context, but css_set_lock
+  * is not irq safe, so bounce to process context.
+  */
+ schedule_work(&cg->delayed_put_work);
+}
+
+/* put_css_set_rcu - helper function to delay a put until after an rcu
```

```
+ * grace period
+ *
+ * free_css_set_rcu can never be called if there are outstanding
+ * put_css_set_rcu calls, so we can reuse cg->rcu_head.
+ */
+static inline void put_css_set_rcu(struct css_set *cg)
+{
+ if (atomic_inc_return(&cg->delayed_put_count) == 1)
+  call_rcu(&cg->rcu_head, _put_css_set_rcu);
+}
+
 /*
  * compare_css_sets - helper function for find_existing_css_set().
  * @cg: candidate css_set being tested
@@ -620,9 +654,11 @@ static struct css_set *find_css_set(
  }

  atomic_set(&res->refcount, 1);
+ atomic_set(&res->delayed_put_count, 0);
  INIT_LIST_HEAD(&res->cg_links);
  INIT_LIST_HEAD(&res->tasks);
  INIT_HLIST_NODE(&res->hlist);
+ INIT_WORK(&res->delayed_put_work, __put_css_set_rcu);

  /* Copy the set of subsystem state objects generated in
   * find_existing_css_set() */
@@ -725,9 +761,9 @@ static struct cgroup *task_cgroup_from_root(struct task_struct *task,
  * cgroup_attach_task(), which overwrites one tasks cgroup pointer with
  * another.  It does so using cgroup_mutex, however there are
  * several performance critical places that need to reference
- * task->cgroup without the expense of grabbing a system global
+ * task->cgroups without the expense of grabbing a system global
  * mutex.  Therefore except as noted below, when dereferencing or, as
- * in cgroup_attach_task(), modifying a task'ss cgroup pointer we use
+ * in cgroup_attach_task(), modifying a task's cgroups pointer we use
  * task_lock(), which acts on a spinlock (task->alloc_lock) already in
  * the task_struct routinely used for such matters.
  *
@@ -1802,8 +1838,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
    ss->attach(ss, cgrp, oldcgrp, tsk, false);
  }
  set_bit(CGRP_RELEASABLE, &oldcgrp->flags);
- synchronize_rcu();
- put_css_set(cg);
+ put_css_set_rcu(cg);

  /*
   * wake up rmdir() waiter. the rmdir should fail since the cgroup
```

```
@@ -3900,6 +3935,7 @@ int __init cgroup_init_early(void)
  INIT_LIST_HEAD(&init_css_set.cg_links);
  INIT_LIST_HEAD(&init_css_set.tasks);
  INIT_HLIST_NODE(&init_css_set.hlist);
+ INIT_WORK(&init_css_set.delayed_put_work, __put_css_set_rcu);
  css_set_count = 1;
  init_cgroup_root(&rootnode);
  root_count = 1;
@@ -4827,9 +4863,9 @@ static u64 current_css_set_refcount_read(struct cgroup *cont,
 {
  u64 count;

- rcu_read_lock();
+ task_lock(current);
  count = atomic_read(&current->cgroups->refcount);
- rcu_read_unlock();
+ task_unlock(current);
  return count;
 }

@@ -4838,12 +4874,10 @@ static int current_css_set_cg_links_read(struct cgroup *cont,
       struct seq_file *seq)
 {
  struct cg_cgroup_link *link;
- struct css_set *cg;

  read_lock(&css_set_lock);
- rcu_read_lock();
- cg = rcu_dereference(current->cgroups);
- list_for_each_entry(link, &cg->cg_links, cg_link_list) {
+ task_lock(current);
+ list_for_each_entry(link, &current->cgroups->cg_links, cg_link_list) {
   struct cgroup *c = link->cgrp;
   const char *name;

@@ -4854,7 +4888,7 @@ static int current_css_set_cg_links_read(struct cgroup *cont,
   seq_printf(seq, "Root %d group %s\n",
       c->root->hierarchy_id, name);
  }
- rcu_read_unlock();
+ task_unlock(current);
  read_unlock(&css_set_lock);
  return 0;
 }
--
1.7.3.1
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs