Subject: [PATCH] cgroup: Remove RCU from task->cgroups
Posted by Colin Cross on Sun, 21 Nov 2010 02:00:24 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 already
use task_lock() or cgroup_lock() to protect against updates,
and only the CGROUP_DEBUG files have RCU read-side critical
sections.

This patch replaces rcu_read_lock() with task_lock(current)
around the debug file acceses to current->cgroups and removes
the synchronize_rcu call in cgroup_attach_task.

Signed-off-by: Colin Cross <ccross@android.com>
---
kernel/cgroup.c | 22 ++++++++----------
1 files changed, 8 insertions(+), 14 deletions(-)

```
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 66a416b..4a40183 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -725,14 +725,11 @@ 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.
- *
- * P.S.  One more locking exception.  RCU is used to guard the
- * update of a tasks cgroup pointer by cgroup_attach_task()
  */

 /**
@@ -1786,7 +1783,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
   retval = -ESRCH;
   goto out;
  }
- rcu_assign_pointer(tsk->cgroups, newcg);
+ tsk->cgroups = newcg;
  task_unlock(tsk);
```

```
  /* Update the css_set linked lists if we're using them */
@@ -1802,7 +1799,6 @@ 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);

   /*
@@ -4827,9 +4823,9 @@ 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 +4834,10 @@ 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 +4848,7 @@ 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

Subject: Re: [PATCH] cgroup: Remove RCU from task->cgroups
Posted by Colin Cross on Sun, 21 Nov 2010 23:02:28 GMT
View Forum Message <> Reply to Message

On Sat, Nov 20, 2010 at 6:00 PM, Colin Cross <ccross@android.com> wrote:
> The synchronize_rcu call in cgroup_attach_task can be very
> expensive. All fastpath accesses to task->cgroups already
> use task_lock() or cgroup_lock() to protect against updates,
> and only the CGROUP_DEBUG files have RCU read-side critical
> sections.
>
> This patch replaces rcu_read_lock() with task_lock(current)
> around the debug file acceses to current->cgroups and removes
> the synchronize_rcu call in cgroup_attach_task.
>
> Signed-off-by: Colin Cross <ccross@android.com>
> ---
> kernel/cgroup.c | 22 ++++++++----------
> 1 files changed, 8 insertions(+), 14 deletions(-)
>

This patch isn't correct, there's an rcu_dereference I missed inside
task_group(), and that's the important one.

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

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
```

Subject: Re: [PATCH] cgroup: Convert synchronize_rcu to call_rcu in cgroup_attach_task

Posted by Colin Cross on Tue, 23 Nov 2010 08:58:39 GMT

View Forum Message <> Reply to Message

On Tue, Nov 23, 2010 at 12:14 AM, Li Zefan <lizf@cn.fujitsu.com> wrote:
> 12:06, Colin Cross wrote:

>> 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.
>>
>
> Other cgroup subsystems also use rcu_read_lock to access task->cgroups,
> for example net_cls cgroup and device cgroup.
I believe the same comment applies as sched.c, I'll update the commit message.

> I don't think the performance of task attaching is so critically
> important that we have to use call_rcu() instead of synchronize_rcu()?
On my desktop, moving a task between cgroups averages 100 ms, and on
an Tegra2 SMP ARM platform it takes 20 ms. Moving a task with many
threads can take hundreds of milliseconds or more. With this patch it
takes 50 microseconds to move one task, a 400x improvement.

>> 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().
>>
>
> What for? What do we gain from doing this for those debug
> interfaces?
Left over from the previous patch that incorrectly dropped RCU
completely. I'll put the rcu_read_locks back.

>> 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(-)
>

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

---

Subject: Re: [PATCH] cgroup: Convert synchronize_rcu to call_rcu in cgroup_attach_task
Posted by Colin Cross on Tue, 23 Nov 2010 20:22:45 GMT
View Forum Message <> Reply to Message

On Tue, Nov 23, 2010 at 12:58 AM, Colin Cross <ccross@android.com> wrote:
> On Tue, Nov 23, 2010 at 12:14 AM, Li Zefan <lizf@cn.fujitsu.com> wrote:
>> 12:06, Colin Cross wrote:
>>> 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.
>>>
>>
>> Other cgroup subsystems also use rcu_read_lock to access task->cgroups,
>> for example net_cls cgroup and device cgroup.
> I believe the same comment applies as sched.c, I'll update the commit message.
>
>> I don't think the performance of task attaching is so critically
>> important that we have to use call_rcu() instead of synchronize_rcu()?
> On my desktop, moving a task between cgroups averages 100 ms, and on
> an Tegra2 SMP ARM platform it takes 20 ms. Moving a task with many
> threads can take hundreds of milliseconds or more. With this patch it
> takes 50 microseconds to move one task, a 400x improvement.
>
>>> 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().
>>>
>>
>> What for? What do we gain from doing this for those debug
>> interfaces?
> Left over from the previous patch that incorrectly dropped RCU
> completely. I'll put the rcu_read_locks back.
>
>>> 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(-)
>>
>

This patch has another problem - calling put_css_set_rcu twice before
an rcu grace period has elapsed would not guarantee the appropriate
rcu grace period for the second call. I'll try a new approach, moving
the parts of put_css_set that need to be protected by rcu into
free_css_set_rcu.

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

---

Subject: Re: [PATCH] cgroup: Convert synchronize_rcu to call_rcu in cgroup_attach_task
Posted by Colin Cross on Wed, 24 Nov 2010 02:10:58 GMT
View Forum Message <> Reply to Message

On Tue, Nov 23, 2010 at 6:06 PM, Li Zefan <lizf@cn.fujitsu.com> wrote:
> Paul Menage wrote:

>> On Sun, Nov 21, 2010 at 8:06 PM, Colin Cross <ccross@android.com> wrote:
>>> 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.
>>
>> I definitely agree with the goal of using lighter-weight
>> synchronization than the current synchronize_rcu() call. However,
>> there are definitely some subtleties to worry about in this code.
>>
>> One of the reasons originally for the current synchronization was to
>> avoid the case of calling subsystem destroy() callbacks while there
>> could still be threads with RCU references to the subsystem state. The
>> fact that synchronize_rcu() was called within a cgroup_mutex critical
>> section meant that an rmdir (or any other significant cgrooup
>> management action) couldn't possibly start until any RCU read sections
>> were done.
>>
>> I suspect that when we moved a lot of the cgroup teardown code from
>> cgroup_rmdir() to cgroup_diput() (which also has a synchronize_rcu()
>> call in it) this restriction could have been eased, but I think I left
>> it as it was mostly out of paranoia that I was missing/forgetting some
>> crucial reason for keeping it in place.
>>
>> I'd suggest trying the following approach, which I suspect is similar
>> to what you were suggesting in your last email
>>
>> 1) make find_existing_css_set ignore css_set objects with a zero refcount
>> 2) change __put_css_set to be simply
>>
>> if (atomic_dec_and_test(&cg->refcount)) {
>>   call_rcu(&cg->rcu_head, free_css_set_rcu);
>> }
>
> If we do this, it's not anymore safe to use get_css_set(), which just
> increments the refcount without checking if it's zero.

I used an alternate approach, removing the css_set from the hash table
in put_css_set, but delaying the deletion to free_css_set_rcu. That
way, nothing can get another reference to the css_set to call
get_css_set on.

Subject: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Colin Cross on Wed, 24 Nov 2010 05:37:03 GMT
View Forum Message <> Reply to Message

Changes the meaning of CGRP_RELEASABLE to be set on any cgroup
that has ever had a task or cgroup in it, or had css_get called
on it. The bit is set in cgroup_attach_task, cgroup_create,
and __css_get. It is not necessary to set the bit in
cgroup_fork, as the task is either in the root cgroup, in
which can never be released, or the task it was forked from
already set the bit in croup_attach_task.

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

```
---
 include/linux/cgroup.h |  12 +--------
 kernel/cgroup.c        |  54 ++++++++++++++++++++---------------------------
 2 files changed, 25 insertions(+), 41 deletions(-)

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index ed4ba11..9e13078 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -84,12 +84,6 @@ enum {
  CSS_REMOVED, /* This CSS is dead */
 };

-/* Caller must verify that the css is not for root cgroup */
-static inline void __css_get(struct cgroup_subsys_state *css, int count)
-{
- atomic_add(count, &css->refcnt);
-}
-
 /*
  * Call css_get() to hold a reference on the css; it can be used
  * for a reference obtained via:
@@ -97,6 +91,7 @@ static inline void __css_get(struct cgroup_subsys_state *css, int count)
  * - task->cgroups for a locked task
  */

+extern void __css_get(struct cgroup_subsys_state *css, int count);
 static inline void css_get(struct cgroup_subsys_state *css)
 {
  /* We don't need to reference count the root state */
@@ -143,10 +138,7 @@ static inline void css_put(struct cgroup_subsys_state *css)
 enum {
  /* Control Group is dead */
  CGRP_REMOVED,
- /*
-  * Control Group has previously had a child cgroup or a task,
```

```
-  * but no longer (only if CGRP_NOTIFY_ON_RELEASE is set)
-  */
+ /* Control Group has ever had a child cgroup or a task */
   CGRP_RELEASABLE,
   /* Control Group requires release notifications to userspace */
   CGRP_NOTIFY_ON_RELEASE,
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 66a416b..34e855e 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -338,7 +338,15 @@ static void free_css_set_rcu(struct rcu_head *obj)
   * compiled into their kernel but not actually in use */
  static int use_task_css_set_links __read_mostly;

-static void __put_css_set(struct css_set *cg, int taskexit)
+/*
+ * refcounted get/put for css_set objects
+ */
+static inline void get_css_set(struct css_set *cg)
+{
+ atomic_inc(&cg->refcount);
+}
+
+static void put_css_set(struct css_set *cg)
  {
   struct cg_cgroup_link *link;
   struct cg_cgroup_link *saved_link;
@@ -364,12 +372,8 @@ static void __put_css_set(struct css_set *cg, int taskexit)
    struct cgroup *cgrp = link->cgrp;
    list_del(&link->cg_link_list);
    list_del(&link->cgrp_link_list);
-   if (atomic_dec_and_test(&cgrp->count) &&
-       notify_on_release(cgrp)) {
-    if (taskexit)
-     set_bit(CGRP_RELEASABLE, &cgrp->flags);
+   if (atomic_dec_and_test(&cgrp->count))
     check_for_release(cgrp);
-   }

    kfree(link);
   }
@@ -379,24 +383,6 @@ static void __put_css_set(struct css_set *cg, int taskexit)
  }

  /*
- * refcounted get/put for css_set objects
- */
-static inline void get_css_set(struct css_set *cg)
```

```
-{
- atomic_inc(&cg->refcount);
-}
-
-static inline void put_css_set(struct css_set *cg)
-{
- __put_css_set(cg, 0);
-}
-
-static inline void put_css_set_taskexit(struct css_set *cg)
-{
- __put_css_set(cg, 1);
-}
-
-/*
  * compare_css_sets - helper function for find_existing_css_set().
  * @cg: candidate css_set being tested
  * @old_cg: existing css_set for a task
@@ -1801,7 +1787,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
   if (ss->attach)
    ss->attach(ss, cgrp, oldcgrp, tsk, false);
  }
- set_bit(CGRP_RELEASABLE, &oldcgrp->flags);
+ set_bit(CGRP_RELEASABLE, &cgrp->flags);
  synchronize_rcu();
  put_css_set(cg);

@@ -3427,6 +3413,8 @@ static long cgroup_create(struct cgroup *parent, struct dentry *dentry,
  if (err < 0)
   goto err_remove;

+ set_bit(CGRP_RELEASABLE, &parent->flags);
+
  /* The cgroup directory was pre-locked for us */
  BUG_ON(!mutex_is_locked(&cgrp->dentry->d_inode->i_mutex));

@@ -3645,7 +3633,6 @@ again:
  cgroup_d_remove_dir(d);
  dput(d);

- set_bit(CGRP_RELEASABLE, &parent->flags);
  check_for_release(parent);

  /*
@@ -4240,7 +4227,7 @@ void cgroup_exit(struct task_struct *tsk, int run_callbacks)
  tsk->cgroups = &init_css_set;
  task_unlock(tsk);
  if (cg)
```

```
-  put_css_set_taskexit(cg);
+  put_css_set(cg);
 }

 /**
@@ -4410,6 +4397,14 @@ static void check_for_release(struct cgroup *cgrp)
 }

 /* Caller must verify that the css is not for root cgroup */
+void __css_get(struct cgroup_subsys_state *css, int count)
+{
+ atomic_add(count, &css->refcnt);
+ set_bit(CGRP_RELEASABLE, &css->cgroup->flags);
+}
+EXPORT_SYMBOL_GPL(__css_get);
+
+/* Caller must verify that the css is not for root cgroup */
 void __css_put(struct cgroup_subsys_state *css, int count)
 {
  struct cgroup *cgrp = css->cgroup;
@@ -4417,10 +4412,7 @@ void __css_put(struct cgroup_subsys_state *css, int count)
  rcu_read_lock();
  val = atomic_sub_return(count, &css->refcnt);
  if (val == 1) {
-  if (notify_on_release(cgrp)) {
-   set_bit(CGRP_RELEASABLE, &cgrp->flags);
-   check_for_release(cgrp);
-  }
+  check_for_release(cgrp);
   cgroup_wakeup_rmdir_waiter(cgrp);
  }
  rcu_read_unlock();
--
1.7.3.1
```

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

---

Subject: [PATCH 2/2] cgroup: Remove call to synchronize_rcu in cgroup_attach_task

Posted by Colin Cross on Wed, 24 Nov 2010 05:37:04 GMT

View Forum Message <> Reply to Message

synchronize_rcu can be very expensive, averaging 100 ms in
some cases. In cgroup_attach_task, it is used to prevent

a task->cgroups pointer dereferenced in an RCU read side
critical section from being invalidated, by delaying the
call to put_css_set until after an RCU grace period.

To avoid the call to synchronize_rcu, make the put_css_set
call rcu-safe by moving the deletion of the css_set links
into free_css_set_work, scheduled by the rcu callback
free_css_set_rcu.

The decrement of the cgroup refcount is no longer
synchronous with the call to put_css_set, which can result
in the cgroup refcount staying positive after the last call
to cgroup_attach_task returns. To allow the cgroup to be
deleted with cgroup_rmdir synchronously after
cgroup_attach_task, have rmdir check the refcount of all
associated css_sets. If cgroup_rmdir is called on a cgroup
for which the css_sets all have refcount zero but the
cgroup refcount is nonzero, reuse the rmdir waitqueue to
block the rmdir until free_css_set_work is called.

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

```
---
 include/linux/cgroup.h |    1 +
 kernel/cgroup.c        |  120 +++++++++++++++++++++++++++++-------------------
 2 files changed, 74 insertions(+), 47 deletions(-)

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index 9e13078..49fdff0 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -279,6 +279,7 @@ struct css_set {

  /* For RCU-protected deletion */
  struct rcu_head rcu_head;
+ struct work_struct work;
 };

 /*
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 34e855e..e752c83 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -267,6 +267,33 @@ static void cgroup_release_agent(struct work_struct *work);
 static DECLARE_WORK(release_agent_work, cgroup_release_agent);
 static void check_for_release(struct cgroup *cgrp);

+/*
+ * A queue for waiters to do rmdir() cgroup. A tasks will sleep when
```

```
+ * cgroup->count == 0 && list_empty(&cgroup->children) && subsys has some
+ * reference to css->refcnt. In general, this refcnt is expected to goes down
+ * to zero, soon.
+ *
+ * CGRP_WAIT_ON_RMDIR flag is set under cgroup's inode->i_mutex;
+ */
+DECLARE_WAIT_QUEUE_HEAD(cgroup_rmdir_waitq);
+
+static void cgroup_wakeup_rmdir_waiter(struct cgroup *cgrp)
+{
+ if (unlikely(test_and_clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags)))
+  wake_up_all(&cgroup_rmdir_waitq);
+}
+
+void cgroup_exclude_rmdir(struct cgroup_subsys_state *css)
+{
+ css_get(css);
+}
+
+void cgroup_release_and_wakeup_rmdir(struct cgroup_subsys_state *css)
+{
+ cgroup_wakeup_rmdir_waiter(css->cgroup);
+ css_put(css);
+}
+
 /* Link structure for associating css_set objects with cgroups */
 struct cg_cgroup_link {
  /*
@@ -326,10 +353,35 @@ static struct hlist_head *css_set_hash(struct cgroup_subsys_state
*css[])
  return &css_set_table[index];
 }

+static void free_css_set_work(struct work_struct *work)
+{
+ struct css_set *cg = container_of(work, struct css_set, work);
+ struct cg_cgroup_link *link;
+ struct cg_cgroup_link *saved_link;
+
+ write_lock(&css_set_lock);
+ list_for_each_entry_safe(link, saved_link, &cg->cg_links,
+     cg_link_list) {
+  struct cgroup *cgrp = link->cgrp;
+  list_del(&link->cg_link_list);
+  list_del(&link->cgrp_link_list);
+  if (atomic_dec_and_test(&cgrp->count)) {
+   check_for_release(cgrp);
+   cgroup_wakeup_rmdir_waiter(cgrp);
```

```
+  }
+  kfree(link);
+ }
+ write_unlock(&css_set_lock);
+
+ kfree(cg);
+}
+
 static void free_css_set_rcu(struct rcu_head *obj)
 {
  struct css_set *cg = container_of(obj, struct css_set, rcu_head);
- kfree(cg);
+
+ INIT_WORK(&cg->work, free_css_set_work);
+ schedule_work(&cg->work);
 }

 /* We don't maintain the lists running through each css_set to its
@@ -348,8 +400,6 @@ static inline void get_css_set(struct css_set *cg)

 static void put_css_set(struct css_set *cg)
 {
- struct cg_cgroup_link *link;
- struct cg_cgroup_link *saved_link;
  /*
   * Ensure that the refcount doesn't hit zero while any readers
   * can see it. Similar to atomic_dec_and_lock(), but for an
@@ -363,21 +413,9 @@ static void put_css_set(struct css_set *cg)
   return;
  }

- /* This css_set is dead. unlink it and release cgroup refcounts */
  hlist_del(&cg->hlist);
  css_set_count--;

- list_for_each_entry_safe(link, saved_link, &cg->cg_links,
-     cg_link_list) {
-  struct cgroup *cgrp = link->cgrp;
-  list_del(&link->cg_link_list);
-  list_del(&link->cgrp_link_list);
-  if (atomic_dec_and_test(&cgrp->count))
-   check_for_release(cgrp);
-
-  kfree(link);
- }
-
  write_unlock(&css_set_lock);
  call_rcu(&cg->rcu_head, free_css_set_rcu);
```

```
 }
@@ -711,9 +749,9 @@ 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.
  *
@@ -895,33 +933,6 @@ static void cgroup_d_remove_dir(struct dentry *dentry)
 }

 /*
- * A queue for waiters to do rmdir() cgroup. A tasks will sleep when
- * cgroup->count == 0 && list_empty(&cgroup->children) && subsys has some
- * reference to css->refcnt. In general, this refcnt is expected to goes down
- * to zero, soon.
- *
- * CGRP_WAIT_ON_RMDIR flag is set under cgroup's inode->i_mutex;
- */
-DECLARE_WAIT_QUEUE_HEAD(cgroup_rmdir_waitq);
-
-static void cgroup_wakeup_rmdir_waiter(struct cgroup *cgrp)
-{
- if (unlikely(test_and_clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags)))
-  wake_up_all(&cgroup_rmdir_waitq);
-}
-
-void cgroup_exclude_rmdir(struct cgroup_subsys_state *css)
-{
- css_get(css);
-}
-
-void cgroup_release_and_wakeup_rmdir(struct cgroup_subsys_state *css)
-{
- cgroup_wakeup_rmdir_waiter(css->cgroup);
- css_put(css);
-}
-
-/*
  * Call with cgroup_mutex held. Drops reference counts on modules, including
  * any duplicate ones that parse_cgroupfs_options took. If this function
  * returns an error, no reference counts are touched.
@@ -1788,7 +1799,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
```

```
    ss->attach(ss, cgrp, oldcgrp, tsk, false);
   }
   set_bit(CGRP_RELEASABLE, &cgrp->flags);
- synchronize_rcu();
+ /* put_css_set will not destroy cg until after an RCU grace period */
   put_css_set(cg);

   /*
@@ -3546,6 +3557,21 @@ static int cgroup_clear_css_refs(struct cgroup *cgrp)
   return !failed;
  }

+/* checks if all of the css_sets attached to a cgroup have a refcount of 0.
+ * Must be called with css_set_lock held */
+static int cgroup_css_sets_empty(struct cgroup *cgrp)
+{
+ struct cg_cgroup_link *link;
+
+ list_for_each_entry(link, &cgrp->css_sets, cgrp_link_list) {
+   struct css_set *cg = link->cg;
+   if (atomic_read(&cg->refcount) > 0)
+     return 0;
+ }
+
+ return 1;
+}
+
  static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
  {
   struct cgroup *cgrp = dentry->d_fsdata;
@@ -3558,7 +3584,7 @@ static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
   /* the vfs holds both inode->i_mutex already */
  again:
   mutex_lock(&cgroup_mutex);
- if (atomic_read(&cgrp->count) != 0) {
+ if (!cgroup_css_sets_empty(cgrp)) {
     mutex_unlock(&cgroup_mutex);
     return -EBUSY;
   }
@@ -3591,7 +3617,7 @@ again:

   mutex_lock(&cgroup_mutex);
   parent = cgrp->parent;
- if (atomic_read(&cgrp->count) || !list_empty(&cgrp->children)) {
+ if (!cgroup_css_sets_empty(cgrp) || !list_empty(&cgrp->children)) {
     clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags);
     mutex_unlock(&cgroup_mutex);
     return -EBUSY;
```

--
1.7.3.1

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

Subject: Re: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Colin Cross on Thu, 25 Nov 2010 00:11:34 GMT
View Forum Message <> Reply to Message

On Wed, Nov 24, 2010 at 3:54 PM, Paul Menage <menage@google.com> wrote:
> On Tue, Nov 23, 2010 at 9:37 PM, Colin Cross <ccross@android.com> wrote:
>> @@ -364,12 +372,8 @@ static void __put_css_set(struct css_set *cg, int taskexit)
>> struct cgroup *cgrp = link->cgrp;
>> list_del(&link->cg_link_list);
>> list_del(&link->cgrp_link_list);
>> - if (atomic_dec_and_test(&cgrp->count) &&
>> - notify_on_release(cgrp)) {
>> - if (taskexit)
>> - set_bit(CGRP_RELEASABLE, &cgrp->flags);
>> + if (atomic_dec_and_test(&cgrp->count))
>> check_for_release(cgrp);
>> - }
>
> We seem to have lost some notify_on_release() checks - maybe move that
> to check_for_release()?
check_for_release immediately calls cgroup_is_releasable, which checks
for the same bit as notify_on_release. There's no need for
CGRP_RELEASABLE to depend on notify_on_release, or to check
notify_on_release before calling check_for_release.

>> /* Caller must verify that the css is not for root cgroup */
>> +void __css_get(struct cgroup_subsys_state *css, int count)
>> +{
>> + atomic_add(count, &css->refcnt);
>> + set_bit(CGRP_RELEASABLE, &css->cgroup->flags);
>> +}
>
> Is css_get() the right place to be putting this? It's not clear to me
> why a subsystem taking a refcount on a cgroup's state should render it
> releasable when it drops that refcount.
I matched the existing behavior, __css_put sets CGRP_RELEASABLE when
refcnt goes to 0.

> Should we maybe clear the CGRP_RELEASABLE flag right before doing the
> userspace callback?
Actually, I think CGRP_RELEASABLE can be dropped entirely.
check_for_release is only called from __css_put, cgroup_rmdir, and
__put_css_set (or free_css_set_work after my second patch). Those all
imply that __css_get, get_css_set, or cgroup_create have been
previously called, which are the functions that set CGRP_RELEASABLE.

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

---

Subject: Re: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Colin Cross on Thu, 25 Nov 2010 00:18:59 GMT
View Forum Message <> Reply to Message

On Wed, Nov 24, 2010 at 4:11 PM, Colin Cross <ccross@android.com> wrote:
> On Wed, Nov 24, 2010 at 3:54 PM, Paul Menage <menage@google.com> wrote:
>> On Tue, Nov 23, 2010 at 9:37 PM, Colin Cross <ccross@android.com> wrote:
>>> @@ -364,12 +372,8 @@ static void __put_css_set(struct css_set *cg, int taskexit)
>>> struct cgroup *cgrp = link->cgrp;
>>> list_del(&link->cg_link_list);
>>> list_del(&link->cgrp_link_list);
>>> - if (atomic_dec_and_test(&cgrp->count) &&
>>> - notify_on_release(cgrp)) {
>>> - if (taskexit)
>>> - set_bit(CGRP_RELEASABLE, &cgrp->flags);
>>> + if (atomic_dec_and_test(&cgrp->count))
>>> check_for_release(cgrp);
>>> - }
>>
>> We seem to have lost some notify_on_release() checks - maybe move that
>> to check_for_release()?
> check_for_release immediately calls cgroup_is_releasable, which checks
> for the same bit as notify_on_release. There's no need for
> CGRP_RELEASABLE to depend on notify_on_release, or to check
> notify_on_release before calling check_for_release.
>
>>> /* Caller must verify that the css is not for root cgroup */
>>> +void __css_get(struct cgroup_subsys_state *css, int count)
>>> +{
>>> + atomic_add(count, &css->refcnt);
>>> + set_bit(CGRP_RELEASABLE, &css->cgroup->flags);
>>> +}
>>
>> Is css_get() the right place to be putting this? It's not clear to me

>> why a subsystem taking a refcount on a cgroup's state should render it
>> releasable when it drops that refcount.
> I matched the existing behavior, __css_put sets CGRP_RELEASABLE when
> refcnt goes to 0.
>
>> Should we maybe clear the CGRP_RELEASABLE flag right before doing the
>> userspace callback?
> Actually, I think CGRP_RELEASABLE can be dropped entirely.
> check_for_release is only called from __css_put, cgroup_rmdir, and
> __put_css_set (or free_css_set_work after my second patch). Those all
> imply that __css_get, get_css_set, or cgroup_create have been
> previously called, which are the functions that set CGRP_RELEASABLE.
Nevermind, that's not true - get_css_set does not set CGRP_RELEASABLE,
cgroup_attach_task does.

If CGRP_RELEASABLE is not cleared before the callback, the
release_agent would be run once when the last task was removed from
the cgroup, and then again if a task failed to be added to the empty
cgroup because the task was exiting, so clearing the flag sounds like
a good idea.

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

---

Subject: Re: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Colin Cross on Fri, 03 Dec 2010 03:07:12 GMT
View Forum Message <> Reply to Message

On Wed, Nov 24, 2010 at 4:21 PM, Paul Menage <menage@google.com> wrote:
> On Wed, Nov 24, 2010 at 4:11 PM, Colin Cross <ccross@android.com> wrote:
>>>
>>> We seem to have lost some notify_on_release() checks - maybe move that
>>> to check_for_release()?
>> check_for_release immediately calls cgroup_is_releasable, which checks
>> for the same bit as notify_on_release. There's no need for
>> CGRP_RELEASABLE to depend on notify_on_release, or to check
>> notify_on_release before calling check_for_release.
>
> OK.
>
>> I matched the existing behavior, __css_put sets CGRP_RELEASABLE when
>> refcnt goes to 0.
>>
>
> Ah, we do appear to have had that behaviour for a while. I don't

> remember the justification for it at this point :-)
>
>> check_for_release is only called from __css_put, cgroup_rmdir, and
>> __put_css_set (or free_css_set_work after my second patch). Those all
>> imply that __css_get, get_css_set, or cgroup_create have been
>> previously called, which are the functions that set CGRP_RELEASABLE.
>
> Not in one case - if we create a new cgroup and try to move a thread
> into it, but the thread is exiting as we move it, we'll call
> put_css_set() on the new css_set, which will drop the refcount on the
> target cgroup back to 0. We wouldn't want the auto-release
> notification to kick in in that situation, I think.

Clearing the CGRP_RELEASABLE bit any time after the tests in
check_for_release introduces a race if __css_get is called between the
check and clearing the bit - the cgroup will have an entry, but the
bit will not be set. Without additional locking in __css_get, I don't
see any way to safely clear CGRP_RELEASABLE.

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

---

Subject: Re: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Colin Cross on Fri, 17 Dec 2010 01:12:42 GMT
View Forum Message <> Reply to Message

On Thu, Dec 16, 2010 at 4:54 PM, Paul Menage <menage@google.com> wrote:
> On Thu, Dec 2, 2010 at 7:07 PM, Colin Cross <ccross@android.com> wrote:
>>> Not in one case - if we create a new cgroup and try to move a thread
>>> into it, but the thread is exiting as we move it, we'll call
>>> put_css_set() on the new css_set, which will drop the refcount on the
>>> target cgroup back to 0. We wouldn't want the auto-release
>>> notification to kick in in that situation, I think.
>>
>> Clearing the CGRP_RELEASABLE bit any time after the tests in
>> check_for_release introduces a race if __css_get is called between the
>> check and clearing the bit - the cgroup will have an entry, but the
>> bit will not be set. Without additional locking in __css_get, I don't
>> see any way to safely clear CGRP_RELEASABLE.
>
> I don't quite follow your argument here. Are you saying that the
> problem is that you could end up spawning a release agent for a cgroup
> that was no longer releasable since it now had a process in it again?
> If so, then I don't think that's a problem - spurious release agent
> invocations for non-empty cgroups will always happen occasionally due

> to races between the kernel and userspace. But a failed move of a task
> into a previously-empty cgroup shouldn't trigger the agent.

No, if you add a new process to the group while check_for_release, the bit could get set by the add for the new process, then cleared by the concurrently running check_for_release. The release agent would be spawned with a process in the group, which is fine, but when RELEASABLE bit would be clear. When the new process was removed, check_for_release would not call the release agent at all.

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

---

Subject: Re: [PATCH 2/2] cgroup: Remove call to synchronize_rcu in cgroup_attach_task
Posted by Bryan Huntsman on Fri, 28 Jan 2011 01:17:26 GMT
View Forum Message <> Reply to Message

On 11/23/2010 09:37 PM, Colin Cross wrote:
> synchronize_rcu can be very expensive, averaging 100 ms in
> some cases. In cgroup_attach_task, it is used to prevent
> a task->cgroups pointer dereferenced in an RCU read side
> critical section from being invalidated, by delaying the
> call to put_css_set until after an RCU grace period.
>
> To avoid the call to synchronize_rcu, make the put_css_set
> call rcu-safe by moving the deletion of the css_set links
> into free_css_set_work, scheduled by the rcu callback
> free_css_set_rcu.
>
> The decrement of the cgroup refcount is no longer
> synchronous with the call to put_css_set, which can result
> in the cgroup refcount staying positive after the last call
> to cgroup_attach_task returns. To allow the cgroup to be
> deleted with cgroup_rmdir synchronously after
> cgroup_attach_task, have rmdir check the refcount of all
> associated css_sets. If cgroup_rmdir is called on a cgroup
> for which the css_sets all have refcount zero but the
> cgroup refcount is nonzero, reuse the rmdir waitqueue to
> block the rmdir until free_css_set_work is called.
>
> Signed-off-by: Colin Cross <ccross@android.com>
> ---
> include/linux/cgroup.h | 1 +
> kernel/cgroup.c | 120 +++++++++++++++++++++++++++++-------------------
> 2 files changed, 74 insertions(+), 47 deletions(-)

>
> diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
> index 9e13078..49fdff0 100644
> --- a/include/linux/cgroup.h
> +++ b/include/linux/cgroup.h
> @@ -279,6 +279,7 @@ struct css_set {
>
>   /* For RCU-protected deletion */
>   struct rcu_head rcu_head;
> + struct work_struct work;
> };
>
> /*
> diff --git a/kernel/cgroup.c b/kernel/cgroup.c
> index 34e855e..e752c83 100644
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -267,6 +267,33 @@ static void cgroup_release_agent(struct work_struct *work);
> static DECLARE_WORK(release_agent_work, cgroup_release_agent);
> static void check_for_release(struct cgroup *cgrp);
>
> +/*
> + * A queue for waiters to do rmdir() cgroup. A tasks will sleep when
> + * cgroup->count == 0 && list_empty(&cgroup->children) && subsys has some
> + * reference to css->refcnt. In general, this refcnt is expected to goes down
> + * to zero, soon.
> + *
> + * CGRP_WAIT_ON_RMDIR flag is set under cgroup's inode->i_mutex;
> + */
> +DECLARE_WAIT_QUEUE_HEAD(cgroup_rmdir_waitq);
> +
> +static void cgroup_wakeup_rmdir_waiter(struct cgroup *cgrp)
> +{
> + if (unlikely(test_and_clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags)))
> + wake_up_all(&cgroup_rmdir_waitq);
> +}
> +
> +void cgroup_exclude_rmdir(struct cgroup_subsys_state *css)
> +{
> + css_get(css);
> +}
> +
> +void cgroup_release_and_wakeup_rmdir(struct cgroup_subsys_state *css)
> +{
> + cgroup_wakeup_rmdir_waiter(css->cgroup);
> + css_put(css);
> +}
> +

> /* Link structure for associating css_set objects with cgroups */
> struct cg_cgroup_link {
>   /*
> @@ -326,10 +353,35 @@ static struct hlist_head *css_set_hash(struct cgroup_subsys_state
*css[])
>   return &css_set_table[index];
> }
>
> +static void free_css_set_work(struct work_struct *work)
> +{
> + struct css_set *cg = container_of(work, struct css_set, work);
> + struct cg_cgroup_link *link;
> + struct cg_cgroup_link *saved_link;
> +
> + write_lock(&css_set_lock);
> + list_for_each_entry_safe(link, saved_link, &cg->cg_links,
> +   cg_link_list) {
> + struct cgroup *cgrp = link->cgrp;
> + list_del(&link->cg_link_list);
> + list_del(&link->cgrp_link_list);
> + if (atomic_dec_and_test(&cgrp->count)) {
> +   check_for_release(cgrp);
> +   cgroup_wakeup_rmdir_waiter(cgrp);
> + }
> + kfree(link);
> + }
> + write_unlock(&css_set_lock);
> +
> + kfree(cg);
> +}
> +
> static void free_css_set_rcu(struct rcu_head *obj)
> {
>   struct css_set *cg = container_of(obj, struct css_set, rcu_head);
> - kfree(cg);
> +
> + INIT_WORK(&cg->work, free_css_set_work);
> + schedule_work(&cg->work);
> }
>
> /* We don't maintain the lists running through each css_set to its
> @@ -348,8 +400,6 @@ static inline void get_css_set(struct css_set *cg)
>
> static void put_css_set(struct css_set *cg)
> {
> - struct cg_cgroup_link *link;
> - struct cg_cgroup_link *saved_link;
>   /*

```
>    * Ensure that the refcount doesn't hit zero while any readers
>    * can see it. Similar to atomic_dec_and_lock(), but for an
> @@ -363,21 +413,9 @@ static void put_css_set(struct css_set *cg)
>     return;
>    }
>
> - /* This css_set is dead. unlink it and release cgroup refcounts */
>    hlist_del(&cg->hlist);
>    css_set_count--;
>
> - list_for_each_entry_safe(link, saved_link, &cg->cg_links,
> -     cg_link_list) {
> -  struct cgroup *cgrp = link->cgrp;
> -  list_del(&link->cg_link_list);
> -  list_del(&link->cgrp_link_list);
> -  if (atomic_dec_and_test(&cgrp->count))
> -   check_for_release(cgrp);
> -
> -  kfree(link);
> - }
> -
>    write_unlock(&css_set_lock);
>    call_rcu(&cg->rcu_head, free_css_set_rcu);
>  }
> @@ -711,9 +749,9 @@ 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.
>    *
> @@ -895,33 +933,6 @@ static void cgroup_d_remove_dir(struct dentry *dentry)
>  }
>
>  /*
> - * A queue for waiters to do rmdir() cgroup. A tasks will sleep when
> - * cgroup->count == 0 && list_empty(&cgroup->children) && subsys has some
> - * reference to css->refcnt. In general, this refcnt is expected to goes down
> - * to zero, soon.
> - *
> - * CGRP_WAIT_ON_RMDIR flag is set under cgroup's inode->i_mutex;
> - */
> -DECLARE_WAIT_QUEUE_HEAD(cgroup_rmdir_waitq);
```

> -
> -static void cgroup_wakeup_rmdir_waiter(struct cgroup *cgrp)
> -{
> - if (unlikely(test_and_clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags)))
> - wake_up_all(&cgroup_rmdir_waitq);
> -}
> -
> -void cgroup_exclude_rmdir(struct cgroup_subsys_state *css)
> -{
> - css_get(css);
> -}
> -
> -void cgroup_release_and_wakeup_rmdir(struct cgroup_subsys_state *css)
> -{
> - cgroup_wakeup_rmdir_waiter(css->cgroup);
> - css_put(css);
> -}
> -
> -/*
> * Call with cgroup_mutex held. Drops reference counts on modules, including
> * any duplicate ones that parse_cgroupfs_options took. If this function
> * returns an error, no reference counts are touched.
> @@ -1788,7 +1799,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
> ss->attach(ss, cgrp, oldcgrp, tsk, false);
> }
> set_bit(CGRP_RELEASABLE, &cgrp->flags);
> - synchronize_rcu();
> + /* put_css_set will not destroy cg until after an RCU grace period */
> put_css_set(cg);
>
> /*
> @@ -3546,6 +3557,21 @@ static int cgroup_clear_css_refs(struct cgroup *cgrp)
> return !failed;
> }
>
> +/* checks if all of the css_sets attached to a cgroup have a refcount of 0.
> + * Must be called with css_set_lock held */
> +static int cgroup_css_sets_empty(struct cgroup *cgrp)
> +{
> + struct cg_cgroup_link *link;
> +
> + list_for_each_entry(link, &cgrp->css_sets, cgrp_link_list) {
> + struct css_set *cg = link->cg;
> + if (atomic_read(&cg->refcount) > 0)
> + return 0;
> + }
> +
> + return 1;

> +}
> +
> static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
> {
>   struct cgroup *cgrp = dentry->d_fsdata;
> @@ -3558,7 +3584,7 @@ static int cgroup_rmdir(struct inode *unused_dir, struct dentry
*dentry)
>   /* the vfs holds both inode->i_mutex already */
> again:
>   mutex_lock(&cgroup_mutex);
> - if (atomic_read(&cgrp->count) != 0) {
> + if (!cgroup_css_sets_empty(cgrp)) {
>   mutex_unlock(&cgroup_mutex);
>   return -EBUSY;
>   }
> @@ -3591,7 +3617,7 @@ again:
>
>   mutex_lock(&cgroup_mutex);
>   parent = cgrp->parent;
> - if (atomic_read(&cgrp->count) || !list_empty(&cgrp->children)) {
> + if (!cgroup_css_sets_empty(cgrp) || !list_empty(&cgrp->children)) {
>   clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags);
>   mutex_unlock(&cgroup_mutex);
>   return -EBUSY;

Tested-by: Mike Bohan <mbohan@codeaurora.org>

I'm responding on Mike's behalf and adding him to this thread. This
patch improves launch time of a test app from ~700ms to ~250ms on MSM,
with much lower variance across tests. We also see UI latency
improvements, but have not quantified the gains.

- Bryan

--
Sent by an employee of the Qualcomm Innovation Center, Inc.
The Qualcomm Innovation Center, Inc. is a member of the Code Aurora Forum.

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

---

Subject: Re: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Bryan Huntsman on Fri, 28 Jan 2011 01:17:32 GMT
View Forum Message <> Reply to Message

On 11/23/2010 09:37 PM, Colin Cross wrote:
> Changes the meaning of CGRP_RELEASABLE to be set on any cgroup
> that has ever had a task or cgroup in it, or had css_get called
> on it. The bit is set in cgroup_attach_task, cgroup_create,
> and __css_get. It is not necessary to set the bit in
> cgroup_fork, as the task is either in the root cgroup, in
> which can never be released, or the task it was forked from
> already set the bit in croup_attach_task.
>
> Signed-off-by: Colin Cross <ccross@android.com>
> ---
> include/linux/cgroup.h | 12 +--------
> kernel/cgroup.c | 54 ++++++++++++++++++++++++++-------------------
> 2 files changed, 25 insertions(+), 41 deletions(-)
>
> diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
> index ed4ba11..9e13078 100644
> --- a/include/linux/cgroup.h
> +++ b/include/linux/cgroup.h
> @@ -84,12 +84,6 @@ enum {
>   CSS_REMOVED, /* This CSS is dead */
> };
>
> -/* Caller must verify that the css is not for root cgroup */
> -static inline void __css_get(struct cgroup_subsys_state *css, int count)
> -{
> - atomic_add(count, &css->refcnt);
> -}
> -
> /*
>   * Call css_get() to hold a reference on the css; it can be used
>   * for a reference obtained via:
> @@ -97,6 +91,7 @@ static inline void __css_get(struct cgroup_subsys_state *css, int count)
>   * - task->cgroups for a locked task
>   */
>
> +extern void __css_get(struct cgroup_subsys_state *css, int count);
> static inline void css_get(struct cgroup_subsys_state *css)
> {
>   /* We don't need to reference count the root state */
> @@ -143,10 +138,7 @@ static inline void css_put(struct cgroup_subsys_state *css)
> enum {
>   /* Control Group is dead */
>   CGRP_REMOVED,
> - /*
> - * Control Group has previously had a child cgroup or a task,
> - * but no longer (only if CGRP_NOTIFY_ON_RELEASE is set)
> - */

```
> + /* Control Group has ever had a child cgroup or a task */
>   CGRP_RELEASABLE,
>   /* Control Group requires release notifications to userspace */
>   CGRP_NOTIFY_ON_RELEASE,
> diff --git a/kernel/cgroup.c b/kernel/cgroup.c
> index 66a416b..34e855e 100644
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -338,7 +338,15 @@ static void free_css_set_rcu(struct rcu_head *obj)
>   * compiled into their kernel but not actually in use */
>  static int use_task_css_set_links __read_mostly;
>
> -static void __put_css_set(struct css_set *cg, int taskexit)
> +/*
> + * refcounted get/put for css_set objects
> + */
> +static inline void get_css_set(struct css_set *cg)
> +{
> + atomic_inc(&cg->refcount);
> +}
> +
> +static void put_css_set(struct css_set *cg)
>  {
>   struct cg_cgroup_link *link;
>   struct cg_cgroup_link *saved_link;
> @@ -364,12 +372,8 @@ static void __put_css_set(struct css_set *cg, int taskexit)
>    struct cgroup *cgrp = link->cgrp;
>    list_del(&link->cg_link_list);
>    list_del(&link->cgrp_link_list);
> -  if (atomic_dec_and_test(&cgrp->count) &&
> -      notify_on_release(cgrp)) {
> -   if (taskexit)
> -    set_bit(CGRP_RELEASABLE, &cgrp->flags);
> +  if (atomic_dec_and_test(&cgrp->count))
>     check_for_release(cgrp);
> -  }
>
>    kfree(link);
>   }
> @@ -379,24 +383,6 @@ static void __put_css_set(struct css_set *cg, int taskexit)
>  }
>
>  /*
> - * refcounted get/put for css_set objects
> - */
> -static inline void get_css_set(struct css_set *cg)
> -{
> - atomic_inc(&cg->refcount);
```

```
> -}
> -
> -static inline void put_css_set(struct css_set *cg)
> -{
> - __put_css_set(cg, 0);
> -}
> -
> -static inline void put_css_set_taskexit(struct css_set *cg)
> -{
> - __put_css_set(cg, 1);
> -}
> -
> -/*
>   * compare_css_sets - helper function for find_existing_css_set().
>   * @cg: candidate css_set being tested
>   * @old_cg: existing css_set for a task
> @@ -1801,7 +1787,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
>    if (ss->attach)
>     ss->attach(ss, cgrp, oldcgrp, tsk, false);
>   }
> - set_bit(CGRP_RELEASABLE, &oldcgrp->flags);
> + set_bit(CGRP_RELEASABLE, &cgrp->flags);
>   synchronize_rcu();
>   put_css_set(cg);
>
> @@ -3427,6 +3413,8 @@ static long cgroup_create(struct cgroup *parent, struct dentry
*dentry,
>   if (err < 0)
>    goto err_remove;
>
> + set_bit(CGRP_RELEASABLE, &parent->flags);
> +
>   /* The cgroup directory was pre-locked for us */
>   BUG_ON(!mutex_is_locked(&cgrp->dentry->d_inode->i_mutex));
>
> @@ -3645,7 +3633,6 @@ again:
>   cgroup_d_remove_dir(d);
>   dput(d);
>
> - set_bit(CGRP_RELEASABLE, &parent->flags);
>   check_for_release(parent);
>
>   /*
> @@ -4240,7 +4227,7 @@ void cgroup_exit(struct task_struct *tsk, int run_callbacks)
>   tsk->cgroups = &init_css_set;
>   task_unlock(tsk);
>   if (cg)
> -  put_css_set_taskexit(cg);
```

```
> +  put_css_set(cg);
>  }
>
>  /**
> @@ -4410,6 +4397,14 @@ static void check_for_release(struct cgroup *cgrp)
>  }
>
>  /* Caller must verify that the css is not for root cgroup */
> +void __css_get(struct cgroup_subsys_state *css, int count)
> +{
> + atomic_add(count, &css->refcnt);
> + set_bit(CGRP_RELEASABLE, &css->cgroup->flags);
> +}
> +EXPORT_SYMBOL_GPL(__css_get);
> +
> +/* Caller must verify that the css is not for root cgroup */
>  void __css_put(struct cgroup_subsys_state *css, int count)
>  {
>   struct cgroup *cgrp = css->cgroup;
> @@ -4417,10 +4412,7 @@ void __css_put(struct cgroup_subsys_state *css, int count)
>   rcu_read_lock();
>   val = atomic_sub_return(count, &css->refcnt);
>   if (val == 1) {
> -  if (notify_on_release(cgrp)) {
> -   set_bit(CGRP_RELEASABLE, &cgrp->flags);
> -   check_for_release(cgrp);
> -  }
> +  check_for_release(cgrp);
>    cgroup_wakeup_rmdir_waiter(cgrp);
>   }
>   rcu_read_unlock();
```

Tested-by: Mike Bohan <mbohan@codeaurora.org>

I'm responding on Mike's behalf and adding him to this thread. This patch improves launch time of a test app from ~700ms to ~250ms on MSM, with much lower variance across tests. We also see UI latency improvements, but have not quantified the gains.

- Bryan

Subject: Re: [PATCH 1/2] cgroup: Set CGRP_RELEASABLE when adding to a cgroup
Posted by Michael Bohan on Fri, 28 Jan 2011 01:48:01 GMT
View Forum Message <> Reply to Message

On 1/27/2011 5:30 PM, Paul Menage wrote:
> On Thu, Jan 27, 2011 at 5:17 PM, Bryan Huntsman<bryanh@codeaurora.org> wrote:
>>
>> Tested-by: Mike Bohan<mbohan@codeaurora.org>
>>
>> I'm responding on Mike's behalf and adding him to this thread. This
>> patch improves launch time of a test app from ~700ms to ~250ms on MSM,
>> with much lower variance across tests. We also see UI latency
>> improvements, but have not quantified the gains.
>>
>
> Is this attached to the wrong patch? I'd thought that it was the other
> patch (removing the rcu_synchronize()) that's the performance booster.
> This one is more about preserving the semantics of the notification
> API.

You are correct. "[PATCH 2/2] cgroup: Remove call to synchronize_rcu in cgroup_attach_task" improved the performance.

To be more correct, I tested this patch (eg. "cgroup: Set CGRP_RELEASABLE when adding to a cgroup") to the degree that it didn't appear to cause any stability or functional regressions when performing the simple benchmark procedure described above. I did also test "[PATCH 2/2] cgroup: Remove call to synchronize_rcu in cgroup_attach_task" independently of this patch to verify that it alone improved the performance.

Thanks,
Mike

--
Employee of Qualcomm Innovation Center, Inc.
Qualcomm Innovation Center, Inc. is a member of Code Aurora Forum

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