Subject: [RFC PATCH 5/5] refresh VM committed space after a task migration
Posted by Andrea Righi on Mon, 09 Jun 2008 23:33:03 GMT
View Forum Message <> Reply to Message

Update VM committed space statistics when a task is migrated from a cgroup to another. To implement this feature we must keep track of the space committed by each task (that is directly accounted in the task_struct).

Signed-off-by: Andrea Righi <righi.andrea@gmail.com>
---
include/linux/sched.h | 3 +++
kernel/fork.c | 3 +++
mm/memcontrol.c | 6 ++++++
3 files changed, 12 insertions(+), 0 deletions(-)

```
diff --git a/include/linux/sched.h b/include/linux/sched.h
index ae0be3c..8b458df 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1277,6 +1277,9 @@ struct task_struct {
  /* cg_list protected by css_set_lock and tsk->alloc_lock */
  struct list_head cg_list;
 #endif
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ atomic_long_t vm_committed_space;
+#endif
 #ifdef CONFIG_FUTEX
  struct robust_list_head __user *robust_list;
 #ifdef CONFIG_COMPAT
diff --git a/kernel/fork.c b/kernel/fork.c
index eaffa56..9fafbdb 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -219,6 +219,9 @@ static struct task_struct *dup_task_struct(struct task_struct *orig)
  /* One for us, one for whoever does the "release_task()" (usually parent) */
  atomic_set(&tsk->usage,2);
  atomic_set(&tsk->fs_excl, 0);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ atomic_long_set(&tsk->vm_committed_space, 0);
+#endif
 #ifdef CONFIG_BLK_DEV_IO_TRACE
  tsk->btrace_seq = 0;
 #endif
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index e3e34e9..bc4923e 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -1334,6 +1334,7 @@ static void mem_cgroup_move_task(struct cgroup_subsys *ss,
```

```
 {
  struct mm_struct *mm;
  struct mem_cgroup *mem, *old_mem;
+ long committed;

  if (mem_cgroup_subsys.disabled)
   return;
@@ -1355,6 +1356,11 @@ static void mem_cgroup_move_task(struct cgroup_subsys *ss,
  if (!thread_group_leader(p))
   goto out;

+ preempt_disable();
+ committed = atomic_long_read(&p->vm_committed_space);
+ atomic_long_sub(committed, &old_mem->vmacct.vm_committed_space);
+ atomic_long_add(committed, &mem->vmacct.vm_committed_space);
+ preempt_enable();
 out:
  mmput(mm);
 }
--
1.5.4.3
```

Subject: Re: [RFC PATCH 5/5] refresh VM committed space after a task migration
Posted by Dave Hansen on Tue, 10 Jun 2008 17:41:28 GMT
View Forum Message <> Reply to Message

On Tue, 2008-06-10 at 01:33 +0200, Andrea Righi wrote:

```
> +       preempt_disable();
> +       committed = atomic_long_read(&p->vm_committed_space);
> +       atomic_long_sub(committed, &old_mem->vmacct.vm_committed_space);
> +       atomic_long_add(committed, &mem->vmacct.vm_committed_space);
> +       preempt_enable();
>  out:
>         mmput(mm);
>  }
```

Why bother with the preempt stuff here? What does the actually protect against? I assume that you're trying to keep other tasks that might run on this CPU from seeing weird, inconsistent numbers in here. Is there some other looks that keeps *other* cpus from seeing this?

In any case, I think it needs a big, fat comment.

-- Dave

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

---

Subject: Re: [RFC PATCH 5/5] refresh VM committed space after a task migration
Posted by Andrea Righi on Wed, 11 Jun 2008 10:37:57 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Tue, 2008-06-10 at 01:33 +0200, Andrea Righi wrote:
>> +	preempt_disable();
>> +	committed = atomic_long_read(&p->vm_committed_space);
>> +	atomic_long_sub(committed, &old_mem->vmacct.vm_committed_space);
>> +	atomic_long_add(committed, &mem->vmacct.vm_committed_space);
>> +	preempt_enable();
>> out:
>> 	mmput(mm);
>> }
>
> Why bother with the preempt stuff here? What does the actually protect
> against? I assume that you're trying to keep other tasks that might run
> on this CPU from seeing weird, inconsistent numbers in here. Is there
> some other looks that keeps *other* cpus from seeing this?
>
> In any case, I think it needs a big, fat comment.

Yes, true, mem_cgroup_move_task() is called after the task->cgroups
pointer has been changed. So, even if task changes its committed space
between the atomic_long_sub() and atomic_long_add() it will be correctly
accounted in the new cgroup.

-Andrea

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

---