Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Paul Menage on Wed, 11 Jun 2008 07:17:31 GMT
View Forum Message <> Reply to Message

On Thu, Jun 5, 2008 at 6:52 PM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
```
> Move Usage at Task Move (just an experimantal for discussion)
> I tested this but don't think bug-free.
>
> In current memcg, when task moves to a new cg, the usage remains in the old cg.
> This is considered to be not good.
```

Is it really such a big deal if we don't transfer the page ownerships
to the new cgroup? As this thread has shown, it's a fairly painful
operation to support. It would be good to have some concrete examples
of cases where this is needed.

```
>
> This is a trial to move "usage" from old cg to new cg at task move.
> Finally, you'll see the problems we have to handle are failure and rollback.
>
> This one's Basic algorithm is
>
>     0. can_attach() is called.
>     1. count movable pages by scanning page table. isolate all pages from LRU.
>     2. try to create enough room in new memory cgroup
>     3. start moving page accouing
>     4. putback pages to LRU.
>     5. can_attach() for other cgroups are called.
>
> A case study.
>
>  group_A -> limit=1G, task_X's usage= 800M.
>  group_B -> limit=1G, usage=500M.
>
> For moving task_X from group_A to group_B.
>  - group_B  should be reclaimed or have enough room.
>
> While moving task_X from group_A to group_B.
>  - group_B's memory usage can be changed
>  - group_A's memory usage can be changed
>
>  We accounts the resouce based on pages. Then, we can't move all resource
>  usage at once.
>
>  If group_B has no more room when we've moved 700M of task_X to group_B,
>  we have to move 700M of task_X back to group_A. So I implemented roll-back.
```

> But other process may use up group_A's available resource at that point.
>
> For avoiding that, preserve 800M in group_B before moving task_X means that
> task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)

I think that pre-reserving in B would be the cleanest solution, and
would save the need to provide rollback.

> 2. Don't move any usage at task move. (current implementation.)
>   Pros.
>   - no complication in the code.
>   Cons.
>   - A task's usage is chareged to wrong cgroup.
>   - Not sure, but I believe the users don't want this.

I'd say stick with this unless there a strong arguments in favour of
changing, based on concrete needs.

>
> One reasone is that I think a typical usage of memory controller is
> fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.

Exactly - this is a good reason *not* to implement move - because then
you drag all the usage of the middleware daemon into the new cgroup.

```
> Index: temp-2.6.26-rc2-mm1/include/linux/cgroup.h
> ===================================================================
> --- temp-2.6.26-rc2-mm1.orig/include/linux/cgroup.h
> +++ temp-2.6.26-rc2-mm1/include/linux/cgroup.h
> @@ -299,6 +299,8 @@ struct cgroup_subsys {
>                        struct cgroup *cgrp, struct task_struct *tsk);
>       void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
>                      struct cgroup *old_cgrp, struct task_struct *tsk);
> +     void (*attach_rollback)(struct cgroup_subsys *ss,
> +                             struct task_struct *tsk);
>       void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
>       void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
>       int (*populate)(struct cgroup_subsys *ss,
> Index: temp-2.6.26-rc2-mm1/kernel/cgroup.c
> ===================================================================
> --- temp-2.6.26-rc2-mm1.orig/kernel/cgroup.c
> +++ temp-2.6.26-rc2-mm1/kernel/cgroup.c
> @@ -1241,7 +1241,7 @@ int cgroup_attach_task(struct cgroup *cg
>               if (ss->can_attach) {
>                       retval = ss->can_attach(ss, cgrp, tsk);
>                       if (retval)
> -                             return retval;
> +                             goto rollback;
>               }
>       }
>
> @@ -1278,6 +1278,13 @@ int cgroup_attach_task(struct cgroup *cg
>       synchronize_rcu();
>       put_css_set(cg);
>       return 0;
> +
> +rollback:
> +     for_each_subsys(root, ss) {
> +             if (ss->attach_rollback)
> +                     ss->attach_rollback(ss, tsk);
> +     }
> +     return retval;
>  }
>
```

I really need to get round to my plan for implementing transactional attach - I've just been swamped by internal stuff recently. Essentially, I think that we need the ability for a subsystem to request either a commit or a rollback following an attach. The big difference to what we have now is that the each subsystem will be able to synchronize itself with the updates to its state pointer in the task's css_set. Also, we need to not be calling attach_rollback on subsystems that didn't get an attach() call.

Paul