Subject: Re: [RFC] Transactional CGroup task attachment
Posted by KAMEZAWA Hiroyuki on Fri, 11 Jul 2008 00:17:45 GMT
View Forum Message <> Reply to Message

Thank you for your effort.

On Wed, 9 Jul 2008 23:46:33 -0700
"Paul Menage" <menage@google.com> wrote:
> 3) memory
>
> Curently the memory cgroup only uses the mm->owner's cgroup at charge
> time, and keeps a reference to the cgroup on the page. However,
> patches have been proposed that would move all non-shared (page count
> == 1) pages to the destination cgroup when the mm->owner moves to a
> new cgroup. Since it's not possible to prevent page count changes
> without locking all mms on the system, even this transaction approach
> can't really give guarantees. However, something like the following
> would probably be suitable. It's very similar to the memrlimit
> approach, except for the fact that we have to handle the fact that the
> number of pages we finally move might not be exactly the same as the
> number of pages we thought we'd be moving.
>
> prepare_attach_sleep() {
> down_read(&mm->mmap_sem);
> if (mm->owner != state->task) return 0;
> count = count_unshared_pages(mm);
> // save the count charged to the new cgroup
> state->subsys[memcgroup_subsys_id] = (void *)count;
> if ((ret = res_counter_charge(state->dest, count)) {
> up_read(&mm->mmap_sem);
> }
> return ret;
> }
>
> commit_attach() {
> if (mm->owner == state->task) {
> final_count = move_unshared_pages(mm, state->dest);
> res_counter_uncharge(state->src, final_count);
> count = state->subsys[memcgroup_subsys_id];
> res_counter_force_charge(state->dest, final_count - count);
> }
> up_read(&mm->mmap_sem);
> }
>
> abort_attach_sleep() {
> if (mm->owner == state->task) {
> count = state->subsys[memcgroup_subsys_id];
> res_counter_uncharge(state->dest, count);

> }
> up_read(&mm->mmap_sem);
> }
>

At frist look, maybe works well. we need some special codes (to move resource)
but that's all.

My small concern is a state change between prepare_attach_sleep() ->
commit_attach(). Hmm...but as you say, we cannot do down_write(mmap_sem).
Maybe inserting some check codes to mem_cgroup_charge() to stop charge while
move is the last thing we can do.

Anyway, if unwinding is supported officially, I think we can find a way
to go.

Thanks,
-Kame

> 4) numtasks:
>
> Numtasks is different from the two memory-related controllers in that
> it may need to move charges from multiple source cgroups (for
> different threads); the memory cgroups only have to deal with the mm
> of a thread-group leader, and all threads in an attach operation are
> from the same thread_group. So numtasks has to be able to handle
> uncharging multiple source cgroups in the commit_attach() operation.
> In order to do this, it requires additional state:
>
> struct numtasks_attach_state {
> int count;
> struct cgroup *cg;
> struct numtasks_attach_state *next;
> }
>
> It will build a list of numtasks_attach_state objects, one for each
> distinct source cgroup; in the general case either there will only be
> a single thread moving or else all the threads in the thread group
> will belong to the same cgroup, in which case this list will only be a
> single element; the list is very unlikely to get to more than a small
> number of elements.
>
> The prepare_attach_sleep() function can rely on the fact that although
> tasks can fork/exit concurrently with the attach, since cgroup_mutex
> is held, no tasks can change cgroups, and therefore a complete list of
> source cgroups can be constructed.
>

```
> prepare_attach_sleep() {
>   for each thread being moved:
>     if the list doesn't yet have an entry for thread->cgroup:
>       allocate new entry with cg = thread->cgroup, count = 0;
>       add new entry to list
>   store list in state->subsys[numtasks_subsys_id];
>   return 0;
> }
>
> Then prepare_attach_nosleep() can move counts under protection of
> tasklist_lock, to prevent any forks/exits
>
> prepare_attach_nosleep() {
>   read_lock(&tasklist_lock);
>   for each thread being moved {
>     find entry for thread->cgroup in list
>     entry->count++;
>     total_count++;
>   }
>   if ((ret = res_counter_charge(state->dest, total_count) != 0) {
>     read_unlock(&tasklist_lock);
>   }
>   return ret;
> }
>
> commit_attach() {
>   for each entry in list {
>     res_counter_uncharge(entry->cg, entry->count);
>   }
>   read_unlock(&tasklist_lock);
>   free list;
> }
>
> abort_attach_nosleep() {
>   // just needs to clear up prepare_attach_nosleep()
>   res_counter_uncharge(state->dest, total_count);
>   read_unlock(&tasklist_lock);
> }
>
> abort_attach_sleep() {
>   // just needs to clean up the list allocated in prepare_attach_sleep()
>   free list;
> }
>
>
> So, thoughts? Is this just way to complex? Have I missed something
> that means this approach can't work?
>
```

> Paul
>

_____

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