Subject: Re: [PATCH] task containersv11 add tasks file interface fix for cpusets
Posted by Paul Jackson on Thu, 11 Oct 2007 23:15:59 GMT
View Forum Message <> Reply to Message

David wrote:
> you could protect the whole thing by sched_hotcpu_mutex, which is
> expressly designed for migrations.
>
> Something like this:
>
> struct cgroup_iter it;
> struct task_struct *p, **tasks;
> int i = 0;
>
> cgroup_iter_start(cs->css.cgroup, &it);
> while ((p = cgroup_iter_next(cs->css.cgroup, &it))) {
>   get_task_struct(p);
>   tasks[i++] = p;
> }
> cgroup_iter_end(cs->css.cgroup, &it);
>
> while (--i >= 0) {
>   sched_migrate_task(tasks[i], cs->cpus_allowed);
>   put_task_struct(tasks[i]);
> }
>
> kernel/sched.c:
>
> void sched_migrate_task(struct task_struct *task,
>   cpumask_t cpus_allowed)
> {
>   mutex_lock(&sched_hotcpu_mutex);
>   set_cpus_allowed(task, cpus_allowed);
>   mutex_unlock(&sched_hotcpu_mutex);
> }

Hmmm ... I hadn't noticed that sched_hotcpu_mutex before.

I wonder what it is guarding? As best as I can guess, it seems, at
least in part, to be keeping the following two items consistent:
 1) cpu_online_map
 2) the per-task cpus_allowed masks

That is, it seems to ensure that a task is allowed to run on some
online CPU.

If that's approximately true, then shouldn't I take sched_hotcpu_mutex

around the entire chunk of code that handles updating a cpusets 'cpus',
from the time it verifies that the requested CPUs are online, until the
time that every affected task has its cpus_allowed updated?

Furthermore, I should probably guard changes to and verifications
against the top_cpuset's cpus_allowed with this mutex as well, as it is
supposed to be a copy of cpu_online_map.

And since all descendent cpusets have to have 'cpus' masks that are
subsets of their parents, this means guarding other chunks of cpuset
code that depend on the consistency of various per-cpuset cpus_allowed
masks and cpu_online_map.

My current intuition is that this expanded use of sched_hotcpu_mutex in
the cpuset code involving various cpus_allowed masks would be a good
thing.

In sum, perhaps sched_hotcpu_mutex is guarding the dispersed kernel
state that depends on what CPUs are online. This includes the per-task
and per-cpuset cpus_allowed masks, all of which are supposed to be some
non-empty subset of the online CPUs.

Taking and dropping the sched_hotcpu_mutex for each task, just around
the call to set_cpus_allowed(), as you suggested above, doesn't seem to
accomplish much that I can see, and certainly doesn't seem to guard the
consistency of cpu_online_map with the tasks cpus_allowed masks.

... lurkers beware ... good chance I haven't a friggin clue ;).
    In other words: Thirty minutes ago I couldn't even spell
    sched_hotcpu_mutex, and now I'm pontificating on it ;);).

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401