Subject: Re: [RFC] cpuset update_cgroup_cpus_allowed
Posted by Paul Jackson on Tue, 16 Oct 2007 09:16:26 GMT
View Forum Message <> Reply to Message

David wrote:
> Why can't you just add a helper function to sched.c:
>
> void set_hotcpus_allowed(struct task_struct *task,
>   cpumask_t cpumask)
> {
>   mutex_lock(&sched_hotcpu_mutex);
>   set_cpus_allowed(task, cpumask);
>   mutex_unlock(&sched_hotcpu_mutex);
> }
>
> And then change each task's cpus_allowed via that function instead of
> set_cpus_allowed() directly?

I guess this would avoid race conditions within the set_cpus_allowed()
routine, between its code to read the cpu_online_map and set the tasks
cpus_allowed ... though if that's useful, don't we really need to add
locking/unlocking on sched_hotcpu_mutex right inside the
set_cpus_allowed() routine, for all users of set_cpus_allowed ??

But I don't see where the above code helps at all deal with the
races I considered in my previous message:

> My solution may be worse than that. Because set_cpus_allowed() will
> fail if asked to set a non-overlapping cpumask, my solution could never
> terminate. If asked to set a cpusets cpus to something that went off
> line right then, this I'd guess this code could keep looping forever,
> looking for cpumasks that didn't match, and then not noticing that it
> was failing to set them so as they would match.

These races involve reading the tasks cpuset cpus_allowed mask, reading
the online map, and both reading and writing the tasks task_struct
cpus_allowed. Unless one holds the relevant lock for the entire
interval surrounding the critical accesses to these values, it won't do
any good that I can see. Just briefly holding a lock around each
separate access is useless.

--

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