Subject: Re: [PATCH] task containersv11 add tasks file interface fix for cpusets
Posted by Paul Jackson on Wed, 03 Oct 2007 20:16:32 GMT
View Forum Message <> Reply to Message

Paul M wrote:
> Given that later in cpusets.txt you say:
>
> >If hotplug functionality is used
> >to remove all the CPUs that are currently assigned to a cpuset,
> >then the kernel will automatically update the cpus_allowed of all
> >tasks attached to CPUs in that cpuset to allow all CPUs
>
> why can't the same thing be done when changing the 'cpus' file manually.
>
> What's wrong with, in update_cpumask(), doing a loop across all
> members of the cgroup and updating their cpus_allowed fields?
>
> The existing cpusets API is broken, since a new child can always be
> forked between reading the tasks file and doing the writes.

Hmmm ... interesting. Yes - that race, between fork and rewriting
the tasks file is there, by design, and has been there for years.

So far as I know, it is a benign race. In practice, no one that I know
has tripped over it. This doesn't imply that we should not look for
opportunities to fix it, but it does mean it is not an urgent fix, in
my experience.

But there may be more issues here than that:
1) The above cpusets.txt text might be both incorrect (at first
glance now, I suspect that it is the cpuset cpus_allowed that
is updated on hotplug events, not the tasks cpus_allowed) and
perhaps stale as well (given some recent cpuset hotplug patches
being worked by Cliff Wickman.)
2) It is not immediately clear to me how the hotplug can work at
present - if it updates just the cpusets cpus_allowed but doesn't
get to the tasks cpus_allowed.
3) So far as I can recall, the only kernel code path to update a
tasks cpus_allowed goes via the "set_cpus_allowed()" routine,
which the kernel/cpuset.c code only calls from one place, that
being the code that handles writing a pid to a cpuset tasks file.
This provides more evidence that (2) above is a problem -- no
hotplug code paths in the cpuset code seem to invoke this key
"set_cpus_allowed()" routine.

Vedy vedy interesting. Like lifting up a rock and seeing a bunch
of weird creepy crawlies under it.

I'm wondering how to proceed from here, in two regards:

1) The cc list on this thread is long and not well focused for the detailed discussion that it may take to unravel this.
2) I am not optimistic that we can unravel this in time for the rapidly approaching start of 2.6.24.

At present, the only one of the bugs or potential bugs noted here that are actually biting me is the bug this patch addressed; the current (yes, knowingly broken) cpuset code requires the attach code to be run even when reattaching to the same cgroup/cpuset to which the task is already attached.

If the 2.6.24 kernel releases with this bug, such that reattaching tasks to their current cpuset becomes a no-op, that will hurt.

Does my patch (the one opening this thread) cause any problems that you know of, Paul M?

So far as I can tell, this patch just removes a minor optimization, and other than its allowing a hack in the cpuset code to persist (a side affect of what logically should be a no-op), this patch should be unnoticeable to user land.

If that's so, can we allow this patch to proceed, likely for the 2.6.24 kernel series, while we work on killing off all the creepy crawlies that your good questions have exposed?

And I'm tempted to continue this discussion on a new thread, as it seems that it could be an involved discussion not well focused on the interests of many on the current cc list of this message. There may be more hotplug interest, and less container interest, than currently seen on this threads cc list.

Any further suggestions, or embarrassing (;) questions?

Thanks!

--

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