Subject: Re: [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by tong.n.li on Fri, 25 May 2007 17:14:58 GMT
View Forum Message <> Reply to Message

On Fri, 2007-05-25 at 21:44 +0530, Srivatsa Vaddagiri wrote:
> >
> > That assumes per-user scheduling groups; other configurations would
> > make it one step for each level of hierarchy. It may be possible to
> > reduce those steps to only state transitions that change weightings
> > and incremental updates of task weightings. By and large, one needs
> > the groups to determine task weightings as opposed to hierarchically
> > scheduling, so there are alternative ways of going about this, ones
> > that would even make load balancing easier.
>
> Yeah I agree that providing hierarchical group-fairness at the cost of single
> (or fewer) scheduling levels would be a nice thing to target for,
> although I don't know of any good way to do it. Do you have any ideas
> here? Doing group fairness in a single level, using a common rb-tree for tasks
> from all groups is very difficult IMHO. We need atleast two levels.
>
> One possibility is that we recognize deeper hierarchies only in user-space,
> but flatten this view from kernel perspective i.e some user space tool
> will have to distributed the weights accordingly in this flattened view
> to the kernel.

Nice work, Vatsa. When I wrote the DWRR algorithm, I flattened the
hierarchies into one level, so maybe that approach can be applied to
your code as well. What I did is to maintain task and task group weights
and reservations separately from the scheduler, while the scheduler only
sees one system-wide weight per task and does not concern about which
group a task is in. The key here is the system-wide weight of each task
should represent an equivalent share to the share represented by the
group hierarchies. To do this, the scheduler looks up the task and group
weights/reservations it maintains, and dynamically computes the
system-wide weight *only* when it need a weight for a given task while
scheduling. The on-demand weight computation makes sure the cost is
small (constant time). The computation itself can be seen from an
example: assume we have a group of two tasks and the group's total share
is represented by a weight of 10. Inside the group, let's say the two
tasks, P1 and P2, have weights 1 and 2. Then the system-wide weight for
P1 is 10/3 and the weight for P2 is 20/3. In essence, this flattens
weights into one level without changing the shares they represent.

tong