Subject: Re: [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Srivatsa Vaddagiri on Fri, 25 May 2007 16:14:24 GMT
View Forum Message <> Reply to Message

On Wed, May 23, 2007 at 11:03:16AM -0700, William Lee Irwin III wrote:
> Well, SMP load balancing is what makes all this hard.

Agreed. I am optimistic that we can achieve good degree of SMP
fairness using similar mechanism as smpnice ..

> On Wed, May 23, 2007 at 10:18:59PM +0530, Srivatsa Vaddagiri wrote:
> > Salient points which needs discussion:
> > 1. This patch reuses CFS core to achieve fairness at group level also.
> > To make this possible, CFS core has been abstracted to deal with generic
> > schedulable "entities" (tasks, users etc).
>
> The ability to handle deeper hierarchies would be useful for those
> who want such semantics.

sure, although the more levels of hierarchy scheduler recoginizes, more
the (accounting/scheduling) cost is!

> On Wed, May 23, 2007 at 10:18:59PM +0530, Srivatsa Vaddagiri wrote:
> > 2. The per-cpu rb-tree has been split to be per-group per-cpu.
> > schedule() now becomes two step on every cpu : pick a group first (from
> > group rb-tree) and a task within that group next (from that group's task
> > rb-tree)
>
> 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.

> On Wed, May 23, 2007 at 10:18:59PM +0530, Srivatsa Vaddagiri wrote:

> > 3. Grouping mechanism - I have used 'uid' as the basis of grouping for
> > timebeing (since that grouping concept is already in mainline today).
> > The patch can be adapted to a more generic process grouping mechanism
> > (like http://lkml.org/lkml/2007/4/27/146) later.
>
> I'd like to see how desirable the semantics achieved by reflecting
> more of the process hierarchy structure in the scheduler groupings are.
> Users, sessions, pgrps, and thread_groups would be the levels of
> hierarchy there, where some handling of orphan pgrps is needed.

Good point. Essentially all users should get fair cpu first, then all sessions/pgrps under a user should get fair share, followed by process-groups under a session, followed by processes in a process-group, followed by threads in a process (phew) .. ?

The container patches by Paul Menage at http://lkml.org/lkml/2007/4/27/146 provide a generic enough mechanism to group tasks in a hierarchical manner for each resource controller. For ex: for the cpu controller, if the desired fairness is as per the above scheme (user/session/pgrp/threads etc), then it is possible to write a script which creates such a tree under cpu controller filesystem:

```
 # mkdir /dev/cpuctl
 # mount -t container -o cpuctl none /dev/cpuctl
```

/dev/cpuctl is the cpu controller filesystem which can look like this:

```
 /dev/cpuctl
  |----uid root
  |       |-- sid 10
  | |      |------ pgrp 20
  |  |      |    |-- process 100
  | |      |          |-- process 101
  | |      |          |
  |       |
  | |-- sid 11
  |
  |--- uid guest
```

(If the cpu controller really supports those many levels that is!)

user scripts can be written to modify this filesystem tree upon every login/session/user creation (if that is possible to trap on). Essentially it lets this semantics (what you ask) be dynamic/tunable by user.

> Kernel compiles are markedly poor benchmarks. Try lat_ctx from lmbench,
> VolanoMark, AIM7, OAST, SDET, and so on.

Thanks for this list of tests. I intend to run all of them if possible
for my next version.

--
Regards,
vatsa

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