Subject: Re: [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Ingo Molnar on Wed, 23 May 2007 18:32:52 GMT
View Forum Message <> Reply to Message

* Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:

> Here's an attempt to extend CFS (v13) to be fair at a group level,
> rather than just at task level. The patch is in a very premature state
> (passes simple tests, smp load balance not supported yet) at this
> point. I am sending it out early to know if this is a good direction
> to proceed.

cool patch! :-)

> 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).

yeah, i like this alot.

The "struct sched_entity" abstraction looks very clean, and that's the main thing that matters: it allows for a design that will only cost us performance if group scheduling is desired.

If you could do a -v14 port and at least add minimal SMP support: i.e. it shouldnt crash on SMP, but otherwise no extra load-balancing logic is needed for the first cut - then i could try to pick all these core changes up for -v15. (I'll let you know about any other thoughts/details when i do the integration.)

> 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)

yeah. It might even become more steps if someone wants to have a different, deeper hierarchy (at the price of performance). Containers will for example certainly want to use one more level.

> 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.

yeah, agreed.

> Some results below, obtained on a 4way (with HT) Intel Xeon box. All
> number are reflective of single CPU performance (tests were forced to
> run on single cpu since load balance is not yet supported).
>
>
>   uid "vatsa"   uid "guest"
>   (make -s -j4 bzImage)   (make -s -j20 bzImage)
>
> 2.6.22-rc1   772.02 sec 497.42 sec (real)
> 2.6.22-rc1+cfs-v13   780.62 sec 478.35 sec (real)
> 2.6.22-rc1+cfs-v13+this patch   776.36 sec 776.68 sec (real)
>
> [An exclusive cpuset containing only one CPU was created and the
> compilation jobs of both users were run simultaneously in this cpuset
>]

looks really promising!

> I also disabled CONFIG_FAIR_USER_SCHED and compared the results with
> cfs-v13:
>
>   uid "vatsa"
>   make -s -j4 bzImage
>
> 2.6.22-rc1+cfs-v13   395.57 sec (real)
> 2.6.22-rc1+cfs-v13+this_patch 388.54 sec (real)
>
> There is no regression I can see (rather some improvement, which I
> can't understand atm). I will run more tests later to check this
> regression aspect.

kernel builds dont really push scheduling micro-costs, rather try
something like 'hackbench.c' to measure that. (kernel builds are of
course one of our primary benchmarks.)

> Request your comments on the future direction to proceed!

full steam ahead please! =B-)

 Ingo