Subject: Re: [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by dev on Fri, 25 May 2007 13:05:16 GMT
View Forum Message <> Reply to Message

Ingo Molnar wrote:
> * Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
>
>
>>Can you repeat your tests with this patch pls? With the patch applied,
>>I am now getting the same split between nice 0 and nice 10 task as
>>CFS-v13 provides (90:10 as reported by top)
>>
>> 5418 guest 20 0 2464 304 236 R 90 0.0 5:41.40 3 hog
>> 5419 guest 30 10 2460 304 236 R 10 0.0 0:43.62 3 nice10hog
>
>
> btw., what are you thoughts about SMP?
>
> it's a natural extension of your current code. I think the best approach
> would be to add a level of 'virtual CPU' objects above struct user. (how
> to set the attributes of those objects is open - possibly combine it
> with cpusets?)

> That way the scheduler would first pick a "virtual CPU" to schedule, and
> then pick a user from that virtual CPU, and then a task from the user.

don't you mean the vice versa:
first use to scheduler, then VCPU (which is essentially a runqueue or rbtree),
then a task from VCPU?

this is the approach we use in OpenVZ and if you don't mind
I would propose to go this way for fair-scheduling in mainstream.
It has it's own advantages and disatvantages.

This is not the easy way to go and I can outline the problems/disadvantages
which appear on this way:
- tasks which bind to CPU mask will bind to virtual CPUs.
  no problem with user tasks, but some kernel threads
  use this to do CPU-related management (like cpufreq).
  This can be fixed using SMP IPI actually.
- VCPUs should no change PCPUs very frequently,
  otherwise there is some overhead. Solvable.

Advantages:
- High precision and fairness.
- Allows to use different group scheduling algorithms
  on top of VCPU concept.
  OpenVZ uses fairscheduler with CPU limiting feature allowing

to set maximum CPU time given to a group of tasks.

> To make group accounting scalable, the accounting object attached to the
> user struct should/must be per-cpu (per-vcpu) too. That way we'd have a
> clean hierarchy like:
>
> CPU #0 => VCPU A [40%] + VCPU B [60%]
> CPU #1 => VCPU C [30%] + VCPU D [70%]

how did you select these 40%:60% and 30%:70% split?

> VCPU A => USER X [10%] + USER Y [90%]
> VCPU B => USER X [10%] + USER Y [90%]
> VCPU C => USER X [10%] + USER Y [90%]
> VCPU D => USER X [10%] + USER Y [90%]
>
> the scheduler first picks a vcpu, then a user from a vcpu. (the actual
> external structure of the hierarchy should be opaque to the scheduler
> core, naturally, so that we can use other hierarchies too)
>
> whenever the scheduler does accounting, it knows where in the hierarchy
> it is and updates all higher level entries too. This means that the
> accounting object for USER X is replicated for each VCPU it participates
> in.

So if 2 VCPUs running on 2 physical CPUs do accounting the have to update the same user X accounting information which is not per-[v]cpu?

> SMP balancing is straightforward: it would fundamentally iterate through
> the same hierarchy and would attempt to keep all levels balanced - i
> abstracted away its iterators already.

Thanks,
Kirill

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