Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Srivatsa Vaddagiri on Fri, 25 May 2007 18:08:50 GMT
View Forum Message <> Reply to Message

On Fri, May 25, 2007 at 08:18:56PM +0400, Kirill Korotaev wrote:
> 2 physical CPUs can't select the same VCPU at the same time.
> i.e. VCPU can be running on 1 PCPU only at the moment.
> and vice versa: PCPU can run only 1 VCPU at the given moment.
>
> So serialization is done when we need to assign VCPU to PCPU moment only,
> not when we select a particular task from the runqueue.
>
> About the contention: you can control how often VCPUs should be rescheduled,
> so the contention can be quite small. This contention is unavoidable in any fair
> scheduler since fairness implies across CPUs accounting and decision making at least
> with some period of time.
>
> Well it is possible to avoid contention at all - if we do fair scheduling
> separately on each CPU. But in this case we still do user-based balancing
> (which requires serialization) and precision can be nasty.

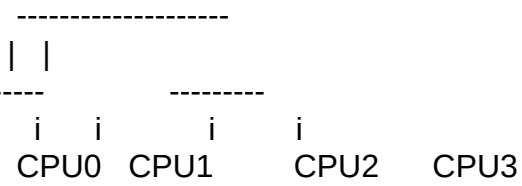
I guess how nasty or not it is depends on interval over which fairness
is expected. Longer the interval, better the scope to provide good fairness
based on load-balance schemes like sched-domain/smpnice ..

> > 2. How would this load balance at virtual cpu level and sched domain based
> > load balancing interact?

[snip]

> load balancing is done taking into account *current* VCPUs assignments to PCPUs.
> i.e. sched domains are taken into account.
> nothing is introduces at schedule() time - not sure what you meant actually by this.
>

Basically, lets say that there are 2 CPUs, each with two threads



CPU0/1 and CPU2/3 are siblings.

Lets say that CPU0 has two or more tasks while CPU[1-3] are idle. Then the HT
optimizations in sched domains based load balancer ensures that CPU2 or
3 picks up one (or more) task from CPU0 rather than CPU1 picking it. The same

logic works at higher packaging levels (core, package, node? etc).

How would the virtual cpu scheduler achieve such optimizations? My thought was such optimizations (if they have to be retained) has to be introduced at virtual cpu schedule time ..?

Not just this, the sched domain based load balancer has other (nice?) properties that it balances less frequently across nodes than across cpus inside a node. This lets tasks execute longer on same node.

Essentially what I wanted to know was : what will be the role of sched domain based load balancer on top of virtual cpu load balancer? Do you disable sched domain based balancer completely? Or does it balance tasks across virtual cpus? Given that virtual cpus can run anywhere, then it needs to be significantly changed to understand that (for ex: the HT/MC optimizations in sched domain balancer needs to becomes aware of this virtual->physical cpu relationship).

> > 4. VCPU ids (namespace) - is it different for different containers?
>
> yes.

Is this namespace different inside kernel context too?

To illustrate, consider my previous example:

CONTAINER1 => VCPU0 + VCPU1
CONTAINER2 => VCPU2 + VCPU3

Lets say that T1 in CONTAINER1 is running on VCPU0/PCPU0 at the same time that T2 in CONTAINER2 is running on VCPU2/PCPU1. As we discussed earlier, they both can see themselves running on same (virtual) cpu 0. Lets say they make system calls now and what does the kernel see this cpu id as thr' smp_processor_id()? Hopefully it is different for the two threads when they are inside kernel ..

> > For ex: can id's of vcpus belonging to different containers (say VCPU0 and
> > VCPU2), as seen by users thr' vgetcpu/smp_processor_id() that is, be same?
>
> yes.
>
> > If so, then potentially two threads belonging to different users may find
> > that they are running -truly simultaneously- on /same/ cpu 0 (one on
> > VCPU0/PCPU0 and another on VCPU2/PCPU1) which normally isn't possible!
>
> yes. but for user space this has no any implications. You see, there is no way for user space
> to determine whether it is "-truly simultaneously- running on /same/ cpu 0".

Hmm ..what if some user space app maintains per-cpu stuff and (now that
we return same cpu id to both tasks running simultaneously on two different
physical cpus) they collide writing to the the same per-cpu area?

> > This may be ok for containers, with non-overlapping cpu id namespace,
> > but when applied to group scheduling for, say, users, which require a
> > global cpu id namespace, wondering how that would be addressed ..
>
> very simple imho.
> the only way from user space to get some task CPU id is /proc.

I believe there is a syscall (vgetcpu?) in works to get a cpu id.

> All you need is to return *some* value there.
> For example, one can report PCPU id to which VCPU is assigned.

That may break other things. What if task had bound to (virtual) cpu0 and
now its call to vgetcpu returns different values at different times,
based on where that virtual cpu0 was running at that moment!

IMHO introduction of virtual cpu for cases which require global cpu
id space will be a pretty complex thingy (both for user space and for
kernel). Not sure if it is worth the benefits.

> > Why is this not a problem for user tasks? Tasks which bind to different
> > CPUs for performance reason now can find that they are running on same
> > (physical) CPU unknowingly.
>
> if there is no high load - tasks will be running on different PCPUs
> as the author was planning, since VCPUs will get different PCPUs for sure.

Again it depends on the application I guess. Even under high load, it
may be the expectation of the user to run on same cpu for correctness
reasons.

> Otherwise - it is not performance critical, and moreover *controversial* to *fairness*.
>
> Let me provide an example why binding is controversial to fairness.
> Imagine that we have 2 USERs - USER1 and USER2 and 2 CPUs in the system.
> USER1 has 50 tasks binded to CPU0 and 50 tasks binded to CPU1.
> USER2 has 1 task.
>
> Let USER2 to be as important as USER1 is, so these USERs should
> share summary CPU time as 1:1.
> How will it work with your approach?

Good example :) USER2's single task will have to share its CPU with
USER1's 50 tasks (unless we modify the smpnice load balancer to

disregard cpu affinity that is - which I would not prefer to do).

Ingo/Peter, any thoughts here? CFS and smpnice probably is "broken"
with respect to such example as above albeit for nice-based tasks.

--
Regards,
vatsa

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