Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by dev on Fri, 25 May 2007 16:18:56 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Fri, May 25, 2007 at 05:05:16PM +0400, Kirill Korotaev wrote:
>
>>>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 [...]
>
>
> So is this how it looks in OpenVZ?
>
> CONTAINER1 => VCPU0 + VCPU1
> CONTAINER2 => VCPU2 + VCPU3
>
> PCPU0 picks a container first, a vcpu next and then a task in it
> PCPU1 also picks a container first, a vcpu next and then a task in it.

correct.

> Few questions:
>
> 1. Are VCPU runqueues (on which tasks are present) global queues?
>
> That is, let's say that both PCPU0 and PCPU1 pick CONTAINER1 to schedule
> (first level) at the same time and next (let's say) they pick same vcpu
> VCPU0 to schedule (second level). Will the two pcpu's now have to be
> serialized for scanning task to schedule next (third level) within VCPU0
> using a spinlock? Won't that shootup scheduling costs (esp on large
> systems), compared to (local scheduling + balance across cpus once in a
> while, the way its done today)?
> Or do you required that two pcpus don't schedule the same vcpu at the
> same time (the way hypervisors normally work)? Even then I would
> imagine a fair level of contention to be present in second step (pick
> a virtual cpu from a container's list of vcpus).

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.

> 2. How would this load balance at virtual cpu level and sched domain based
> load balancing interact?
>
> The current sched domain based balancing code has many HT/MC/SMT related
> optimizations, which ensure that tasks are spread across physical
> threads/cores/packages in a most efficient manner - so as to utilize
> hardware bandwidth to the maximum. You would now need to introduce
> those optimizations essentially at schedule() time ..? Don't know
> if that is a wise thing to do.

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.

> 3. How do you determine the number of VCPUs per container? Is there any
> relation for number of virtual cpus exposed per user/container and
> the number of available cpus? For ex: in case of user-driven
> scheduling, we would want all users to see the same number of cpus
> (which is the number available in the system).

by default every user is given num_online_cpus() VCPUs, i.e. it can
run on all physical CPUs at the same time. If needed a user can be limited.

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

yes.

> 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".

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

>>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, [...]
>
>
> 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.
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?

>>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.
>

>
> I just don't know if this benefit of high degree of fairness is worth the
> complexity it introduces. Besides having some data which shows how much better
> is is with respect to fairness/overhead when compared with other approaches
> (like smpnice) would help I guess. I will however let experts like Ingo make
> the final call here :)

sure. The "perfect" solution doesn't exist :(So I would be happy to know Ingo
opinion as well.

Thanks,
Kirill

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

---

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.

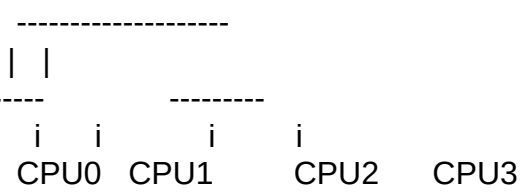
I guess how nasty or not it is depends on interval over which fairness
is expected. Longer the interval, better the scope to provide good fairness
based on load-balance schemes like sched-domain/smpnice ..

> > 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

---

Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Peter Williams on Sat, 26 May 2007 00:17:42 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> 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).

I don't think that ignoring cpu affinity is an option. Setting the cpu
affinity of tasks is a deliberate policy action on the part of the
system administrator and has to be honoured. Load balancing has to do
the best it can in these circumstances which may mean sub optimal
distribution of the load BUT it is result of a deliberate policy
decision by the system administrator.

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

See above. I think that faced with cpu affinity use by the system
administrator that smpnice will tend towards a task to cpu allocation
that is (close to) the best that can be achieved without violating the
cpu affinity assignments. (It may take a little longer than normal but
it should get there eventually.)

You have to assume that the system administrator knows what (s)he's
doing and is willing to accept the impact of their policy decision on
the overall system performance.

Having said that, if it was deemed necessary you could probably increase
the speed at which the load balancer converged on a good result in the
face of cpu affinity by keeping a "pinned weighted load" value for each
run queue and using that to modify find_busiest_group() and
find_busiest_queue() to be a bit smarter. But I'm not sure that it
would be worth the added complexity.

Peter
--
Peter Williams   pwil3058@bigpond.net.au

"Learning, n. The kind of ignorance distinguishing the studious."
  -- Ambrose Bierce

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

---

Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Srivatsa Vaddagiri on Mon, 28 May 2007 17:26:58 GMT
View Forum Message <> Reply to Message

On Sat, May 26, 2007 at 10:17:42AM +1000, Peter Williams wrote:
> I don't think that ignoring cpu affinity is an option. Setting the cpu
> affinity of tasks is a deliberate policy action on the part of the
> system administrator and has to be honoured.

mmm ..but users can set cpu affinity w/o administrator priveleges ..

--
Regards,

vatsa

_____

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

Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Peter Williams on Tue, 29 May 2007 00:18:43 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Sat, May 26, 2007 at 10:17:42AM +1000, Peter Williams wrote:
>> I don't think that ignoring cpu affinity is an option. Setting the cpu
>> affinity of tasks is a deliberate policy action on the part of the
>> system administrator and has to be honoured.
>
> mmm ..but users can set cpu affinity w/o administrator priveleges ..
>

OK. So you have to assume the users know what they're doing. :-)

In reality though, the policy of allowing ordinary users to set affinity on their tasks should be rethought.

In any case, there's no point having cpu affinity if it's going to be ignored. Maybe you could have two levels of affinity: 1. if set by a root it must be obeyed; and 2. if set by an ordinary user it can be overridden if the best interests of the system dictate. BUT I think that would be a bad idea.

Peter
--
Peter Williams pwil3058@bigpond.net.au

"Learning, n. The kind of ignorance distinguishing the studious."
-- Ambrose Bierce

_____

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

Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Peter Williams on Tue, 29 May 2007 03:30:24 GMT
View Forum Message <> Reply to Message

Peter Williams wrote:
> Srivatsa Vaddagiri wrote:
>> On Sat, May 26, 2007 at 10:17:42AM +1000, Peter Williams wrote:
>>> I don't think that ignoring cpu affinity is an option. Setting the
>>> cpu affinity of tasks is a deliberate policy action on the part of
>>> the system administrator and has to be honoured.
>>
>> mmm ..but users can set cpu affinity w/o administrator priveleges ..
>>
>
> OK. So you have to assume the users know what they're doing. :-)
>
> In reality though, the policy of allowing ordinary users to set affinity
> on their tasks should be rethought.

After more contemplation, I now think I may have gone overboard here. I am now of the opinion that any degradation of overall system performance due to the use of cpu affinity would be confined to the tasks with cpu affinity set. So there's no need to prevent ordinary users from setting cpu affinity on their own processes as any degradation will only affect them.

So it goes back to the situation where you have to assume that they know what they're doing and obey their policy.

>
> In any case, there's no point having cpu affinity if it's going to be
> ignored. Maybe you could have two levels of affinity: 1. if set by a
> root it must be obeyed; and 2. if set by an ordinary user it can be
> overridden if the best interests of the system dictate. BUT I think
> that would be a bad idea.

This idea is now not just bad but unnecessary.

Peter
--
Peter Williams pwil3058@bigpond.net.au

"Learning, n. The kind of ignorance distinguishing the studious."
  -- Ambrose Bierce