Subject: Re: [RFC][PATCH 4/6] Fix (bad?) interactions between SCHED_RT and SCHED_NORMAL tasks
Posted by Srivatsa Vaddagiri on Tue, 12 Jun 2007 15:43:32 GMT
View Forum Message <> Reply to Message

On Tue, Jun 12, 2007 at 04:31:38PM +0200, Dmitry Adamushko wrote:
> >But isn't that the same result we would have obtained anyways had we
> >called update_load_fair() on all lrq's on every timer tick? If a user's
> >lrq was inactive for several ticks, then its exec_delta will be seen as
> >zero for those several ticks, which means we would compute its 'this_load'
> >to be
> >zero as well for those several ticks?
>
> Yeah.. seems to be so. But let's consider whether these 'inactive ticks' are
> really inactive [1] :
>
> The fact that user's tasks are not active at the moment of a timer
> interrupt doesn't mean
> they were not active _during_ the last tick.

sure

> That's why another
> approach in update_load_fair() which doesn't depend on a snapshot of
> rq->raw_weighted_load
> at timer tick's time. I guess, we'd lose this with 'inactive ticks',
> right?

Sorry this is not clear. We'd lose what with 'inactive' ticks?

If you are referring to the delta execution time a user's lrq consumed
in the middle of a tick and whether we would lose them during subsequent
update_load(), the answer IMO is no. Becasue put_prev_task_fair() would
account for the small delta execution time when the task/lrq got
descheduled.

> ok, maybe
> it's not that important for per-user cpu_load, duno at the moment.

I would say any lossy accounting would be bad in long run. If you think
put_prev_task_fair()->update_curr() still leaves open some problem, I
would be interested in knowing that.

> >Basically what I want to know is, are we sacrificing any accuracy here
> >because of "deferring" smoothening of cpu_load for a (inactive) lrq
> >(apart from the inaccurate figure used during load_balance as you point
> >out below).
>

> At least, we are getting some inaccuracy (not in a generic case
> though) due to the
>
```
>       if (exec_delta64 > (u64)TICK_NSEC)
>               exec_delta64 = (u64)TICK_NSEC;   [*]
```
>
> in update_load_fair()..

If that is a problem (and I tend to agree that it is), then it is not unique to
group lrq accounting. So we have common problems to solve :)

> and that's smth I want to try changing...

good.

> >Assuming the lrq was inactive for all those 3 ticks and became active at
> >4th tick, would the end result of cpu_load (as obtained in my code) be
> >any different than calling update_load_fair() on all lrq on each tick?
>
> With the current code, yes - it may be. In case, [*] condition (see
> above) comes into play (and these 'inactive' ticks were not really
> inactive as described above).

Yes sure, we need to fix that assumption that exec_delta64 can't be
greater than TICK_NSEC. And I assume you will fix that?

> >If there is any bug in 'replay lost ticks' loop in the patch I posted, then
> >it should already be present in current (i.e v16) implementation of
> >update_load_fair()?
>
> I think, you are right.

good :)

> >Yes, patch #5 introduces group-aware load-balance. It is two-step:
> >
> >First, we identify busiest group and busiest queue, based on
> >rq->raw_weighted_load/cpu_load (which is accumulation of weight from all
> >clases on a CPU). This part of the code is untouched.
>
> I'll take a look (e.g. I guess, we have got a notion of "user's
> weght"... so does/how a user's weight contribute to his tasks weight..

A user's weight controls fraction of CPU the user's tasks as a whole receive.

A task's weight controls fraction of CPU the task will receive -within-
the fraction alloted to that user.

Strictly speaking, a task's weight need not have to depend on its user's
weight. This is true if scheduler core recognizes both user and task levels of
scheduling in the hierarchy. If the scheduler were to recognize fewer
levels of hierarchy, then we will have to take into account a user's
weight in calculation task weight. See thread anchored at
http://lkml.org/lkml/2007/5/26/81 for a description of this idea.

> otherwise, I think, the approach of determining
> the busiest CPU based only on pure tasks' weight would be wrong.. will
> look at it first).

The load considered for determining busiest group/queue is the summation
of -all- task's load on a CPU. That's why I introduced update_load() in
Patch #4 which captures load from real-time tasks as well as
SCHED_NORMAL tasks. When you are changing that update_load() function
(based on class->update_load callback), it would be nice to keep this in
mind (that I need a weight field representing summation of all tasks
weights).

> >> If it's user's lrq :: cpu_load[] .. then it _still_ shows the load at
> >> the moment of T1 while we are at the moment T2 (and user1 was not
> >> active during dT)..
> >
> >Good point. So how do we solve this? I really really want to avoid
> >running update_load_fair() on all lrq's every tick (it will be a massive
> >overhead).
>
> yeahh.. have to think about it.
> btw, I recall the patch #4 adds some light but noticeable overhead,
> right? did you look at where exactly the overhead comes from?

This probably comes from the split up raw_weighted_load/nr_running
fields. Although I don't know if the overhead is that noticeable in
practice. Let me know if you feel any difference with Patch #4 applied!

> > I am assuming that lrqs don't remain inactive for a long time
> >(given CFS's fairness promise!) and hence probably their cpu_load[] also
> >won't be -that- stale in practice?
>
> I guess, it's not only about CFS but about the users' behavior, which
> is something
> we can't control and so can't rely on it.
> Say, a user was active till the moment T1 and then just gone.. - all
> his tasks are really inactive.
> So at the moment T2 user's lrq :: cpu_load will still express the
> situation at the moment T1?
> As long as user's lrq is not involved in 'load balancing', this

> inaccuracy can be revealed only if the info is exported via /proc.
>
> But say, user's task becomes finally active after _a lot_ of inactive
> ticks (the user came back).. now it's in the rq and waiting for its
> turn (which can be easily > 1 tick).. in the mean time 'load
> balancing' is triggered.. and it considers the old lrq :: cpu_load[]

I still think that this 'stale' cpu_load data won't last long enough to
serious affect load balance decisions. But something I agree definitely to
watch for during tests. Thanks for the heads up on this possibility!

> P.S.
>
> just a personal impression.. I'm quite confused by this 'lrq' name...
> it looks pretty similar to 'Irq' (with a big 'i') and I can't stop
> reading it as 'IRQ' [chores: so stop it!]

:-)

> would be smth like 'cfs_rq' or even 'sched_rq' better? :-)

I chose lrq to mean local run queue. Other names I can think of are
entity_rq or ...actually cfs_rq (as you suggest) sounds better. I will
make this change unless Ingo thinks otherwise.

--
Regards,
vatsa
____