Subject: Re: [RFC][PATCH 4/6] Fix (bad?) interactions between SCHED_RT and SCHED_NORMAL tasks
Posted by Dmitry Adamushko on Tue, 12 Jun 2007 14:31:38 GMT
View Forum Message <> Reply to Message

On 12/06/07, Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com> wrote:
> > [...]
> >
> > just substitute {exec,fair}_delta == 1 in the following code:
> >
> > tmp64 = SCHED_LOAD_SCALE * exec_delta64;
> > do_div(tmp64, fair_delta);
> > tmp64 *= exec_delta64;
> > do_div(tmp64, TICK_NSEC);
> > this_load = (unsigned long)tmp64;
> >
> > we'd get
> >
> > tmp64 = 1024 * 1;
> > tmp64 =/ 1;
> > tmp64 *= 1;
> > tmp64 /= 1000000;
> >
> > as a result, this_load = 1024/1000000; which is 0 (no floating point calc.).
>
> Ok ..
>
> 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. 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? ok, maybe
it's not that important for per-user cpu_load, duno at the moment.

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

>
> 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).

> Even though this lost ticks loop is easily triggered with user-based lrqs,
> I think the same "loop" can be seen in current CFS code (i.e say v16)
> when low level timer interrupt handler replays such lost timer ticks (say we
> were in a critical section for some time with timer interrupt disabled).
> As an example see arch/powerpc/kernel/time.c:timer_interrupt() calling
> account_process_time->scheduler_tick in a loop.
>
> 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.

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

> > If it's user's lrq :: cpu_load[] .. then it _still_ shows the load at
> > the moment 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?

> 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[]
...

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? :-)

> --
> Regards,
> vatsa

--

Best regards,
Dmitry Adamushko

_____

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