Subject: Re: [ckrm-tech] [RFC] [PATCH 0/3] Add group fairness to CFS
Posted by Srivatsa Vaddagiri on Thu, 31 May 2007 08:33:53 GMT
View Forum Message <> Reply to Message

On Wed, May 30, 2007 at 11:36:47PM -0700, William Lee Irwin III wrote:
> On Thu, May 31, 2007 at 11:18:28AM +0530, Srivatsa Vaddagiri wrote:
> > Hmm ..the fact that each task runs for a minimum of 1 tick seems to
> > complicate the matters to me (when doing group fairness given a single
> > level hierarchy). A user with 1000 (or more) tasks can be unduly
> > advantaged compared to another user with just 1 (or fewer) task
> > because of this?
>
> Temporarily, yes. All this only works when averaged out.

So essentially when we calculate delta_mine component for each of those
1000 tasks, we will find that it has executed for 1 tick (4 ms say) but
its fair share was very very low.

fair_share = delta_exec * p->load_weight / total_weight

If p->load_weight has been calculated after factoring in hierarchy (as
you outlined in a previous mail), then p->load_weight of those 1000 tasks
will be far less compared to the p->load_weight of one task belonging to
other user, correct? Just to make sure I get all this correct:

User U1 has tasks T0 - T999
User U2 has task T1000

assuming each task's weight is 1 and each user's weight is 1 then:

WT0 = (WU1 / WU1 + WU2) * (WT0 / WT0 + WT1 + ... + WT999)
= (1 / 1 + 1) * (1 / 1000)
= 1/2000
= 0.0005

WT1 ..WT999 will be same as WT0

whereas, weight of T1000 will be:

WT1000 = (WU1 / WU1 + WU2) * (WT1000 / WT1000)
= (1 / 1 + 1) * (1/1)
= 0.5

?

So when T0 (or T1 ..T999) executes for 1 tick (4ms), their fair share would
be:
 T0's fair_share (delta_mine)
   = 4 ms * 0.0005 / (0.0005 * 1000 + 0.5)
   = 4 ms * 0.0005 / 1
   = 0.002 ms (2000 ns)

This would cause T0's ->wait_runtime to go negative sharply, causing it to be
inserted back in rb-tree well ahead in future. One change I can forsee
in CFS is with regard to limit_wait_runtime() ..We will have to change
its default limit, atleast when group fairness thingy is enabled.

Compared to this when T1000 executes for 1 tick, its fair share would be
calculated as:

 T1000's fair_share (delta_mine)
    = 4 ms * 0.5 / (0.0005 * 1000 + 0.5)
    = 4 ms * 0.5 / 1
    = 2 ms (2000000 ns)

Its ->wait_runtime will drop less significantly, which lets it be
inserted in rb-tree much to the left of those 1000 tasks (and which indirectly
lets it gain back its fair share during subsequent schedule cycles).

Hmm ..is that the theory?

Ingo, do you have any comments on this approach?

/me is tempted to try this all out.

> The basic
> idea is that you want a constant upper bound on the difference between
> the CPU time a task receives and the CPU time it was intended to get.
> This discretization is one of the larger sources of the "error" in the
> CPU time granted. The constant upper bound usually only applies to the
> largest difference for any task. When absolute values of differences
> are summed across tasks the aggregate will be O(tasks) because there's
> something almost like a constant per-task lower bound a la Heisenberg.
> It would have to get more exact the more tasks there are on the system
> for that to work, and something of the opposite actually holds.
>
> It might be appropriate for the scheduler to dynamically adjust a
> periodic timer's period or to set up one-shot timers at involuntary
> preemption times in order to achieve more precise fairness in this
> sort of situation. In the case of few preemption points such one-shot
> code or low periodicity code would also save on taking interrupts that
> would otherwise manifest as overhead.

>
> In short, a user with many tasks can reap a temporary advantage
> relative to users with fewer tasks because of this, but over time,
> longer-running tasks will receive the CPU time intended to within
> some constant upper bound, provided other things aren't broken.

--
Regards,
vatsa

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