Subject: Re: [RFC][PATCH 0/6] Add group fairness to CFS - v1
Posted by Ingo Molnar on Tue, 12 Jun 2007 06:26:12 GMT
View Forum Message <> Reply to Message

* Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com> wrote:

> On Mon, Jun 11, 2007 at 09:39:31PM +0200, Ingo Molnar wrote:
> > i mean bit 6, value 64. I flipped around its meaning in -v17-rc4, so the
> > new precise stats code there is now default-enabled - making SMP
> > load-balancing more accurate.
>
> I must be missing something here. AFAICS, cpu_load calculation still
> is timer-interrupt driven in the -v17 snapshot you sent me. Besides,
> there is no change in default value of bit 6 b/n v16 and v17:
>
> -unsigned int sysctl_sched_features __read_mostly = 1 | 2 | 4 | 8 | 0 | 0;
> +unsigned int sysctl_sched_features __read_mostly = 0 | 2 | 4 | 8 | 0 | 0;
>
> So where's this precise stats based calculation of cpu_load?

but there's a change in the interpretation of bit 6:

```
-       if (!(sysctl_sched_features & 64)) {
-               this_load = this_rq->raw_weighted_load;
+       if (sysctl_sched_features & 64) {
+               this_load = this_rq->lrq.raw_weighted_load;
```

the update of the cpu_load[] value is timer interrupt driven, but the _value_ that is sampled is not. Previously we used ->raw_weighted_load (at whatever value it happened to be at the moment the timer irq hit the system), now we basically use a load derived from the fair-time passed since the last scheduler tick. (Mathematically it's close to an integral of load done over that period) So it takes all scheduling activities and all load values into account to calculate the average, not just the value that was sampled by the scheduler tick.

this, besides being more precise (it for example correctly samples short-lived, timer-interrupt-driven workloads too, which were largely 'invisible' to the previous load calculation method), also enables us to make the scheduler tick hrtimer based in the (near) future. (in essence making the scheduler tick-less even when there are tasks running)

> Anyway, do you agree that splitting the cpu_load/nr_running fields so
> that:
>
> rq->nr_running        = total count of -all- tasks in runqueue
> rq->raw_weighted_load   = total weight of -all- tasks in runqueue
> rq->lrq.nr_running    = total count of SCHED_NORMAL/BATCH tasks in runqueue

> rq->lrq.raw_weighted_load = total weight of SCHED_NORMAL/BATCH tasks in runqueue
>
> is a good thing to avoid SCHED_RT<->SCHED_NORMAL/BATCH mixup (as
> accomplished in Patch #4)?

yes, i agree in general, even though this causes some small overhead.
This also has another advantage: the inter-policy isolation and load
balancing is similar to what fair group scheduling does, so even 'plain'
Linux will use the majority of the framework.

> If you don't agree, then I will make this split dependent on
> CONFIG_FAIR_GROUP_SCHED

no, i'd rather avoid that #ifdeffery.

> > > Patch 6 hooks up scheduler with container patches in mm (as an
> > > interface for task-grouping functionality).
>
> Just to be clear, by container patches, I am referring to "process"
> container patches from Paul Menage [1]. They aren't necessarily tied
> to "virtualization-related" container support in -mm tree, although I
> believe that "virtualization-related" container patches will make use
> of the same "process-related" container patches for their
> task-grouping requirements. Phew ..we need better names!

i'd still like to hear back from Kirill & co whether this framework is
flexible enough for their work (OpenVZ, etc.) too.

Ingo

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