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 10:26:22 GMT
View Forum Message <> Reply to Message

On Tue, Jun 12, 2007 at 11:03:36AM +0200, Dmitry Adamushko wrote:
> I had an idea of per-sched-class 'load balance' calculator. So that
> update_load() (as in your patch) would look smth like :
>
> ...
> struct sched_class *class = sched_class_highest;
> unsigned long total = 0;
>
> do {
> 		total += class->update_load(..., now);
> 		class = class->next;
> 	} while (class);
> ...
>
> and e.g. update_load_fair() would become a fair_sched_class ::
> update_load().
>
> That said, all the sched_classes would report a load created by their
> entities (tasks) over the last sampling period. Ideally, the
> calculation should not be merely based on the 'raw_weighted_load' but
> rather done in a similar way to update_load_fair() as in v17.

I like this idea. It neatly segregates load calculation across classes.
It effectively replaces what update_load() function I introduced in
Patch #4.

Btw what will update_load_rt() return?

> > static void entity_tick(struct lrq *lrq, struct sched_entity *curr)
> > {
> > 	struct sched_entity *next;
> > 	struct rq *rq = lrq_rq(lrq);
> > 	u64 now = __rq_clock(rq);
> >
> >+ 	/* replay load smoothening for all ticks we lost */
> >+ 	while (time_after_eq64(now, lrq->last_tick)) {
> >+ 		update_load_fair(lrq);
> >+ 		lrq->last_tick += TICK_NSEC;
> >+ 	}
>
> I think, it won't work properly this way. The first call returns a
> load for last TICK_NSEC and all the consequent ones report zero load
> ('this_load = 0' internally)..

mm ..

```
exec_delta64 = this_lrq->delta_exec_clock + 1;
this_lrq->delta_exec_clock = 0;
```

So exec_delta64 (and fair_delta64) should be min 1 in successive calls. How can that lead to this_load = 0?

The idea behind 'replay lost ticks' is to avoid load smoothening of
-every- lrq -every- tick. Lets say that there are ten lrqs
(corresponding to ten different users). We load smoothen only the currently
active lrq (whose task is currently running). Other lrqs load get smoothened
as soon as they become active next time (thus catching up with all lost ticks).

--
Regards,
vatsa