

---

Subject: Re: [PATCH v3 4/6] add a new scheduler hook for context switch  
Posted by [Peter Zijlstra](#) on Wed, 30 May 2012 11:20:51 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:

> To be able to count the number of switches per-cgroup, and merging  
> Paul's hint that it would be better to do it in fair.c and rt.c,  
> I am introducing a new write-side walk through a new scheduler hook,  
> called at every context switch away from a task (prev).  
>  
> Read-side is greatly simplified, and as I'll show, the performance impact  
> does not seem huge. First, aside from the function call, this walk is  
> O(depth), which is not likely to be huge (if it is, the performance  
> impact is indeed bad - but one can argue this is a good punishment)  
>  
> Also, this walk is likely to be cache-hot, since it is at most the very  
> same loop done by put\_prev\_task, except it loops depth - 1 instead of depth  
> times. This is specially important not to hurt tasks in the root cgroup,  
> that will pay just a branch.

/me cries a little.. I was hoping to fix put\_prev\_task.. see:

<https://lkml.org/lkml/2012/2/16/487>

(I've actually got a 4 patch split out of that if anybody cares)

Its just one of those things stuck behind the -ENOTIME tree :/

The plan is to 'merge' put\_prev\_task and pick\_next\_task into one and avoid a lot of the up-down walking.

You just added a constraint for always having to walk the entire thing up -- cgroups is too damn expensive already, we should be trimming this nonsense not bloating it.

> However, put\_prev\_task is called many times from multiple places,  
> and the logic to differentiate a context switch from another kind of  
> put would make a mess out of it.

I'm hoping the fold of put\_prev in pick\_next as per that patch I referenced could help some, but the cross class switch makes that messy still :/

Reducing the indirect calls is good, adding them is bad.. which makes me the worst offender I'm afraid.

> On a 4-way x86\_64, hackbench -pipe 1 process 4000 shows the following results:  
> - units are seconds to complete the whole benchmark

```

> - percentual stdev for easier assesment
>
> Task sitting in the root cgroup:
> Without patchset: 4.857700 (0.69 %)
> With patchset: 4.863700 (0.63 %)
> Difference : 0.12 %

```

Just increase the repeat count :-)

```
$ perf stat -e cycles --repeat 100 perf bench sched messaging -p -g 100
```

```

48,826,146,710 cycles # 2.470 GHz ( +- 0.17% )
2.149005270 seconds time elapsed ( +- 0.12% )

```

Anyway, a few nits on the below patch..

```

> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 4c1d7e9..db4f2c3 100644
> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -1894,6 +1894,14 @@ fire_sched_out_preempt_notifiers(struct task_struct *curr,
>
> #endif /* CONFIG_PREEMPT_NOTIFIERS */
>
> +static void

```

inline

```

> sched_class_context_switch(struct rq *rq, struct task_struct *prev)
> +{
> +#if defined(CONFIG_FAIR_GROUP_SCHED) || defined(CONFIG_RT_GROUP_SCHED)
> + if (prev->sched_class->context_switch)
> + prev->sched_class->context_switch(rq, prev);
> +#endif
> +}
> +

```

```

> diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
> index 940e6d1..c26fe38 100644
> --- a/kernel/sched/fair.c
> +++ b/kernel/sched/fair.c
> @@ -2993,6 +2993,20 @@ static struct task_struct *pick_next_task_fair(struct rq *rq)
> return p;
> }
>
> +static void context_switch_fair(struct rq *rq, struct task_struct *p)

```

```

> +{
> +#ifdef CONFIG_FAIR_GROUP_SCHED
> + struct cfs_rq *cfs_rq;
> + struct sched_entity *se = &p->se;
> +
> + while (se->parent) {
> + se = se->parent;
> + cfs_rq = group_cfs_rq(se);
> + cfs_rq->nr_switches++;
> + }
> +#endif
> +}
> +

```

Put the whole function inside an existing `#ifdef` block of the right kind.

```

> /*
>  * Account for a descheduled task:
>  */
> @@ -5255,6 +5269,7 @@ const struct sched_class fair_sched_class = {
> .check_preempt_curr = check_preempt_wakeup,
>
> .pick_next_task = pick_next_task_fair,
> + .context_switch = context_switch_fair,

```

Put the `#ifdef` here, so that the method is NULL when `!FAIR_GROUP`, saves an indirect nop call for some weird configs.

```

> .put_prev_task = put_prev_task_fair,
>
> #ifdef CONFIG_SMP

```

idem for `sched/rt.c`