Subject: Re: [PATCH v2 2/5] account guest time per-cgroup as well.
Posted by Paul Turner on Sat, 26 May 2012 04:44:58 GMT
View Forum Message <> Reply to Message

On 04/09/2012 03:25 PM, Glauber Costa wrote:
```
> In the interest of providing a per-cgroup figure of common statistics,
> this patch adds a nr_switches counter to each group runqueue (both cfs
> and rt).
>
> To avoid impact on schedule(), we don't walk the tree at stat gather
> time. This is because schedule() is called much more frequently than
> the tick functions, in which we do walk the tree.
>
> When this figure needs to be read (different patch), we will
> aggregate them at read time.
>
> Signed-off-by: Glauber Costa <glommer-bzQdu9zFT3WakBO8gow8eQ@public.gmane.org>
> ---
>  kernel/sched/core.c  |   32 ++++++++++++++++++++++++++++++++
>  kernel/sched/sched.h |    3 +++
>  2 files changed, 35 insertions(+), 0 deletions(-)
>
> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 1cfb7f0..1ee3772 100644
> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -3168,6 +3168,37 @@ pick_next_task(struct rq *rq)
>  }
>
>  /*
> + * For all other data, we do a tree walk at the time of
> + * gathering. We want, however, to minimize the impact over schedule(),
> + * because... well... it's schedule().
> + *
> + * Therefore we only gather for the current cgroup, and walk the tree
> + * at read time
> + */
> +static void update_switches_task_group(struct rq *rq,
> +          struct task_struct *prev,
> +          struct task_struct *next)
> +{
> +#ifdef CONFIG_CGROUP_SCHED
> + int cpu = cpu_of(rq);
> +
> + if (rq->curr_tg == &root_task_group)
> +  goto out;
> +
> +#ifdef CONFIG_FAIR_GROUP_SCHED
> + if (prev->sched_class == &fair_sched_class)
> +  rq->curr_tg->cfs_rq[cpu]->nr_switches++;
> +#endif
> +#ifdef CONFIG_RT_GROUP_SCHED
> + if (prev->sched_class == &rt_sched_class)
> +  rq->curr_tg->rt_rq[cpu]->nr_switches++;
> +#endif
```

With this approach why differentiate cfs vs rt? These could both just
be on the task_group.

This could then just be

```
if (prev != root_task_group)
  task_group(prev)->nr_switches++;
```

Which you could wrap in a nice static inline that disappears when
CONFIG_CGROUP_PROC_STAT isn't there

Another way to go about this would be to promote (demote?) nr_switches
to the sched_entity. At which point you know you only need to update
yours, and conditionally update your parents.

But.. that's still gross.. Hmm..

```
> +out:
> + rq->curr_tg = task_group(next);
```

If you're going to task_group every time anyway you might as well just
take it against prev -- then you don't have to cache rq->curr_tg?

Another way to do this would be:

```
On cfs_rq, rt_rq add:
  int prev_rq_nr_switches, nr_switches

On put_prev_prev_task_fair (against a task)
  cfs_rq_of(prev->se)->prev_rq_nr_switches = rq->nr_switches

On pick_next_task_fair:
  if (cfs_rq_of(prev->se)->prev_rq_nr_switches != rq->nr_switches)
    cfs_rq->nr_switches++;
```

On aggregating the value for read: +1 if prev_rq_nr_running != rq->nr_running
[And equivalent for sched_rt]

While this is no nicer (and fractionally more expensive but this is
never something we'd enable by default), it at least gets the goop out
of schedule().

```
> +#endif
> +}
> +
> +/*
>   * __schedule() is the main scheduler function.
>   */
>  static void __sched __schedule(void)
> @@ -3230,6 +3261,7 @@ need_resched:
>    rq->curr = next;
>    ++*switch_count;
>
> +  update_switches_task_group(rq, prev, next);
>    context_switch(rq, prev, next); /* unlocks the rq */
>    /*
>     * The context switch have flipped the stack from under us
> diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
> index b8bcd147..3b300f3 100644
> --- a/kernel/sched/sched.h
> +++ b/kernel/sched/sched.h
> @@ -224,6 +224,7 @@ struct cfs_rq {
>
>  #ifdef CONFIG_FAIR_GROUP_SCHED
>   struct rq *rq; /* cpu runqueue to which this cfs_rq is attached */
> + u64 nr_switches;
>
>   /*
>    * leaf cfs_rqs are those that hold tasks (lowest schedulable entity in
> @@ -307,6 +308,7 @@ struct rt_rq {
>   struct rq *rq;
>   struct list_head leaf_rt_rq_list;
>   struct task_group *tg;
> + u64 nr_switches;
>  #endif
>  };
>
> @@ -389,6 +391,7 @@ struct rq {
>   unsigned long nr_uninterruptible;
>
>   struct task_struct *curr, *idle, *stop;
> + struct task_group *curr_tg;
```

This is a little gross. Is task_group even defined without CONFIG_CGROUP_SCHED?

```
>   unsigned long next_balance;
>   struct mm_struct *prev_mm;
>
```