
Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by [akpm](#) on Thu, 01 May 2008 21:00:26 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Thu, 1 May 2008 23:11:06 +0530
Balaji Rao <balajirao@gmail.com> wrote:

```
> This implements a couple of basic statistics for the CPU resource controller.
>
> v2->v3
> -----
> Proper locking while collecting stats. Thanks to Peter Zijlstra for suggesting
> the delta approach.
>
> This applies against 2.6.25-mm1
> ---
>
> Signed-off-by: Balaji Rao <balajirao@gmail.com>
>
> diff --git a/kernel/sched.c b/kernel/sched.c
> index bbdc32a..5bda75a 100644
> --- a/kernel/sched.c
> +++ b/kernel/sched.c
> @@ -248,10 +248,51 @@ struct cfs_rq;
>
> static LIST_HEAD(task_groups);
>
> +#ifdef CONFIG_CGROUP_SCHED
> +
> +#define CPU_CGROUP_STAT_THRESHOLD 1 << 30
```

Needs a comment, and parentheses?

```
> +enum cpu_cgroup_stat_index {
> + CPU_CGROUP_STAT_UTIME, /* Ustime of the task group */
> + CPU_CGROUP_STAT_STIME, /* Kerntime of the task group */
> +
> + CPU_CGROUP_STAT_NSTATS,
> +};
> +
> +struct cpu_cgroup_stat_cpu {
> + s64 count[CPU_CGROUP_STAT_NSTATS];
> + u32 delta[CPU_CGROUP_STAT_NSTATS];
> +} ____cacheline_aligned_in_smp;
```

That's 128 bytes on ia64.

```
> +struct cpu_cgroup_stat {
```

```
> + struct cpu_cgroup_stat_cpu cpustat[NR_CPUS];
> + spinlock_t lock;
> +};
```

And with NR_CPUS=1024, we're starting to talk serious pigginess.

And there's one of these per task_group! Chances are we just won't be able to allocate that much contiguous memory, so that will solve the problem ;)

```
> +/* Called under irq disable. */
> +static void __cpu_cgroup_stat_add(struct cpu_cgroup_stat *stat,
> + enum cpu_cgroup_stat_index idx, int val)
> +{
> + int cpu = smp_processor_id();
> + unsigned long flags;
> +
> + BUG_ON(!irqs_disabled());
```

Ok...

```
> + stat->cpustat[cpu].delta[idx] += val;
> +
> + if (stat->cpustat[cpu].delta[idx] > CPU_CGROUP_STAT_THRESHOLD) {
> + spin_lock_irqsave(&stat->lock, flags);
```

so we could have used plain old spin_lock() here.

```
> + stat->cpustat[cpu].count[idx] += stat->cpustat[cpu].delta[idx];
> + stat->cpustat[cpu].delta[idx] = 0;
> + spin_unlock_irqrestore(&stat->lock, flags);
> + }
> +}
> +#endif
> +
> +/* task group related information */
> + struct task_group {
> + #ifdef CONFIG_CGROUP_SCHED
> + struct cgroup_subsys_state css;
> + struct cpu_cgroup_stat stat;
> + #endif
> +
> + #ifdef CONFIG_FAIR_GROUP_SCHED
> + @@ -3837,6 +3878,16 @@ void account_user_time(struct task_struct *p, cputime_t cputime)
> + cpustat->nice = cputime64_add(cpustat->nice, tmp);
> + else
> + cpustat->user = cputime64_add(cpustat->user, tmp);
> +
> + /* Charge the task's group */
```

```

> + #ifdef CONFIG_CGROUP_SCHED
> + {
> + struct task_group *tg;
> + tg = task_group(p);
> + __cpu_cgroup_stat_add(&tg->stat, CPU_CGROUP_STAT_UTIME,
> + cputime_to_msecs(cputime));
> + }
> + #endif
> }
>
> /*
> @@ -3900,6 +3951,15 @@ void account_system_time(struct task_struct *p, int hardirq_offset,
> cpustat->idle = cputime64_add(cpustat->idle, tmp);
> /* Account for system time used */
> acct_update_integrals(p);
> +
> + #ifdef CONFIG_CGROUP_SCHED
> + {
> + struct task_group *tg;
> + tg = task_group(p);
> + __cpu_cgroup_stat_add(&tg->stat, CPU_CGROUP_STAT_STIME,
> + cputime_to_msecs(cputime));
> + }
> + #endif
> }
>
> /*
> @@ -7838,7 +7898,9 @@ struct task_group *sched_create_group(void)
> }
> list_add_rcu(&tg->list, &task_groups);
> spin_unlock_irqrestore(&task_group_lock, flags);
> -
> + #ifdef CONFIG_CGROUP_SCHED
> + spin_lock_init(&tg->stat.lock);
> + #endif
> return tg;
>
> err:
> @@ -8249,6 +8311,48 @@ static u64 cpu_shares_read_u64(struct cgroup *cgrp, struct cftype
> *cft)
>
> return (u64) tg->shares;
> }
> +
> + static s64 cpu_cgroup_read_stat(struct cpu_cgroup_stat *stat,
> + enum cpu_cgroup_stat_index idx)
> + {
> + int cpu;

```

```

> + s64 ret = 0;
> + unsigned long flags;
> +
> + spin_lock_irqsave(&stat->lock, flags);
> + for_each_possible_cpu(cpu) {
> +   stat->cpustat[cpu].count[idx] += stat->cpustat[cpu].delta[idx];
> +   stat->cpustat[cpu].delta[idx] = 0;
> +   ret += stat->cpustat[cpu].count[idx];
> + }
> + spin_unlock_irqrestore(&stat->lock, flags);
> +
> + return ret;
> +}

```

That loop iterates 1024 times, under spin_lock_irqsave(). On a 2-way. Sad, no?

```

> +static const struct cpu_cgroup_stat_desc {
> + const char *msg;
> + u64 unit;
> +} cpu_cgroup_stat_desc[] = {
> + [CPU_CGROUP_STAT_UTIME] = { "utime", 1, },
> + [CPU_CGROUP_STAT_STIME] = { "stime", 1, },
> +};
> +
> +static int cpu_cgroup_stats_show(struct cgroup *cgrp, struct cftype *cft,
> +   struct cgroup_map_cb *cb)
> +{
> + struct task_group *tg = cgroup_tg(cgrp);
> + struct cpu_cgroup_stat *stat = &tg->stat;
> + int i;
> +
> + for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
> +   s64 val;
> +   val = cpu_cgroup_read_stat(stat, i);
> +   val *= cpu_cgroup_stat_desc[i].unit;
> +   cb->fill(cb, cpu_cgroup_stat_desc[i].msg, val);
> + }
> + return 0;
> +}
> #endif
>
> #ifdef CONFIG_RT_GROUP_SCHED
> @@ -8295,6 +8399,10 @@ static struct cftype cpu_files[] = {
>   .write_u64 = cpu_rt_period_write_uint,
>   },
> #endif
> + {

```

```
> + .name = "stat",  
> + .read_map = cpu_cgroup_stats_show,  
> + },  
> };  
>
```

Rethink, please...

There are numerous reasons here for implementing the counters as dynamically-allocated, per-online-cpu things, with a cpu-hotplug notifier. All a bit of a hassle, but that's life.

Anyway, forget all that.

Did you consider using `include/linux/percpu_counter.h`?

If so, what was wrong with it?

Because it would be much better to fix per-cpu counters than to invent new stuff.

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