Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Paul Turner on Wed, 30 May 2012 11:24:29 GMT
View Forum Message <> Reply to Message

```
On Wed, May 30, 2012 at 2:48 AM, Glauber Costa <glommer@parallels.com> wrote:
> The cpuacct cgroup already exposes user and system numbers in a per-cgroup
> fashion. But they are a summation along the whole group, not a per-cpu figure.
> Also, they are coarse-grained version of the stats usually shown at places
> like /proc/stat.
>
> I want to have enough cgroup data to emulate the /proc/stat interface. To
> achieve that, I am creating a new file "stat_percpu" that displays the
> fine-grained per-cpu data. The original data is left alone.
>
> The format of this file resembles the one found in the usual cgroup's stat
> files. But of course, the fields will be repeated, one per cpu, and prefixed
> with the cpu number.
>
> Therefore, we'll have something like:
>
>  cpu0.user X
>  cpu0.system Y
>  ...
>  cpu1.user X1
>  cpu1.system Y1
>  ...
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
> CC: Paul Turner <pjt@google.com>
> ---
>  kernel/sched/core.c |   33 +++++++++++++++++++++++++++++++++
>  1 file changed, 33 insertions(+)
>
> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 220d416..4c1d7e9 100644
> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -8178,6 +8178,35 @@ static int cpuacct_stats_show(struct cgroup *cgrp, struct cftype *cft,
>        return 0;
>  }
>
> +static inline void do_fill_cb(struct cgroup_map_cb *cb, struct cpuacct *ca,
> +                              char *str, int cpu, int index)
> +{
> +       char name[24];
> +       struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
> +
> +	snprintf(name, sizeof(name), "cpu%d.%s", cpu, str);
> +	cb->fill(cb, name, cputime64_to_clock_t(kcpustat->cpustat[index]));
> +}
> +
> +static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
> +				      struct cgroup_map_cb *cb)
> +{
> +	struct cpuacct *ca = cgroup_ca(cgrp);
> +	int cpu;
> +
> +	for_each_online_cpu(cpu) {
> +		do_fill_cb(cb, ca, "user", cpu, CPUTIME_USER);
> +		do_fill_cb(cb, ca, "nice", cpu, CPUTIME_NICE);
> +		do_fill_cb(cb, ca, "system", cpu, CPUTIME_SYSTEM);
> +		do_fill_cb(cb, ca, "irq", cpu, CPUTIME_IRQ);
> +		do_fill_cb(cb, ca, "softirq", cpu, CPUTIME_SOFTIRQ);
> +		do_fill_cb(cb, ca, "guest", cpu, CPUTIME_GUEST);
> +		do_fill_cb(cb, ca, "guest_nice", cpu, CPUTIME_GUEST_NICE);
> +	}
> +
```

I don't know if there's much that can be trivially done about it but I
suspect these are a bit of a memory allocation time-bomb on a many-CPU
machine. The cgroup:seq_file mating (via read_map) treats everything
as /one/ record. This means that seq_printf is going to end up
eventually allocating a buffer that can fit _everything_ (as well as
every power-of-2 on the way there). Adding insult to injury is that
that the backing buffer is kmalloc() not vmalloc().

200+ bytes per-cpu above really is not unreasonable (46 bytes just for
the text, plus a byte per base 10 digit we end up reporting), but that
then leaves us looking at order-12/13 allocations just to print this
thing when there are O(many) cpus.

```
> +	return 0;
> +}
> +
>  static struct cftype files[] = {
> 	{
> 		.name = "usage",
> @@ -8192,6 +8221,10 @@ static struct cftype files[] = {
> 		.name = "stat",
> 		.read_map = cpuacct_stats_show,
> 	},
> +	{
> +		.name = "stat_percpu",
> +		.read_map = cpuacct_stats_percpu_show,
> +	},
>        { }     /* terminate */
>  };
>
> --
> 1.7.10.2
>
```