
Subject: Re: [-mm] CPU controller statistics (v5)
Posted by [akpm](#) on Wed, 04 Jun 2008 21:44:08 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Wed, 4 Jun 2008 01:35:15 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> From: Balaji Rao <balajirao@gmail.com>
>
> This is a repost of Balaji's patches at
> <http://kerneltrap.org/mailarchive/linux-kernel/2008/5/11/1791504>
> I've made changes to the format exported to user space. I've used
> clock_t, since /proc/<pid>/stat uses the same format to export data.
>
> I've run some basic tests to verify that the patches work.
>
> Andrew could you please consider them for inclusion if there are no
> objections to this patch. This patch helps fill a void in the controllers
> we have w.r.t. statistics
>
> The patch is against the latest git.
>

That is not a changelog. Please always include a (good) changelog with each iteration of a patch.

Were all my previous comments addressed? Most, it seems.

>
> diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
> index e155aa7..60a25cb 100644
> --- a/include/linux/cgroup.h
> +++ b/include/linux/cgroup.h
> @@ -293,6 +293,7 @@ int cgroup_is_descendant(const struct cgroup *cgrp);
> struct cgroup_subsys {
> struct cgroup_subsys_state *(*create)(struct cgroup_subsys *ss,
> struct cgroup *cgrp);
> + void (*initialize)(int early);
> void (*pre_destroy)(struct cgroup_subsys *ss, struct cgroup *cgrp);
> void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cgrp);
> int (*can_attach)(struct cgroup_subsys *ss,
> diff --git a/kernel/cgroup.c b/kernel/cgroup.c
> index 15ac0e1..77569d7 100644
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -2553,6 +2553,9 @@ int __init cgroup_init_early(void)
>
> if (ss->early_init)

```

> cgroup_init_subsys(ss);
> +
> + if (ss->initialize)
> + ss->initialize(1);
> }
> return 0;
> }
> @@ -2577,6 +2580,9 @@ int __init cgroup_init(void)
> struct cgroup_subsys *ss = subsys[i];
> if (!ss->early_init)
> cgroup_init_subsys(ss);
> +
> + if (ss->initialize)
> + ss->initialize(0);

```

Can we avoid these tests? By requiring that cgroup_subsys.initialize() always be non-zero? It might make sense, and it might not...

```

> }
>
> /* Add init_css_set to the hash table */
> diff --git a/kernel/sched.c b/kernel/sched.c
> index bfb8ad8..d6df3d3 100644
> --- a/kernel/sched.c
> +++ b/kernel/sched.c
> @@ -245,11 +245,32 @@ static DEFINE_MUTEX(sched_domains_mutex);
> struct cfs_rq;
>
> static LIST_HEAD(task_groups);
> +ifdef CONFIG_CGROUP_SCHED
> +define CPU_CGROUP_STAT_THRESHOLD (1 << 30)
> +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 {
> + struct percpu_counter cpustat[CPU_CGROUP_STAT_NSTATS];
> +};
> +
> +static void __cpu_cgroup_stat_add(struct cpu_cgroup_stat *stat,
> + enum cpu_cgroup_stat_index idx, s64 val)
> +{
> + if (stat)
> + percpu_counter_add(&stat->cpustat[idx], val);
> +}

```

```

> +#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
> @@ -3885,6 +3906,16 @@ 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) * NSEC_PER_MSEC);
> + }
> +#endif
> }
>
> /*
> @@ -3942,8 +3973,17 @@ void account_system_time(struct task_struct *p, int hardirq_offset,
> cpustat->irq = cputime64_add(cpustat->irq, tmp);
> else if (softirq_count())
> cpustat->softirq = cputime64_add(cpustat->softirq, tmp);
> - else if (p != rq->idle)
> + else if (p != rq->idle) {
> cpustat->system = cputime64_add(cpustat->system, tmp);
> +#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) * NSEC_PER_MSEC);
> + }
> +#endif

```

The above two almost-identical code sequences should, I suggest, be broken out into a standalone helper function, which has two implementations, the CONFIG_CGROUP_SCHED=n version of which is a do-nothing stub, in the usual fashion.

I suggested this last time.

```
> + }
> else if (atomic_read(&rq->nr_iowait) > 0)
>   cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
> else
> @@ -8325,6 +8365,24 @@ unsigned long sched_group_shares(struct task_group *tg)
> }
> #endif
>
> +static s64 cpu_cgroup_read_stat(struct cpu_cgroup_stat *stat,
> + enum cpu_cgroup_stat_index idx)
> +{
> + if (stat)
> + return nsec_to_clock_t(
> +   percpu_counter_read(&stat->cpustat[idx]));
> +
> + return 0;
> +}
```

ick at the code layout. How about this:

```
if (!stat)
    return 0;
return nsec_to_clock_t(percpu_counter_read(&stat->cpustat[idx]));
```

?

```
> +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, },
> +};
> +
> #ifdef CONFIG_RT_GROUP_SCHED
> /*
>  * Ensure that the real time constraints are schedulable.
> @@ -8551,10 +8609,41 @@ static inline struct task_group *cgroup_tg(struct cgroup *cgrp)
>     struct task_group, css);
> }
>
> +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 < CPU_CGROUP_STAT_NSTATS; i++) {
```

Please prefer to put a blank line between end-of-locals and start-of-code. It does make the code somewhat easier to read.

```
> + 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;
> +}
> +
> +static void cpu_cgroup_initialize(int early)
> +{
> + int i;
> + struct cpu_cgroup_stat *stat;
> +
> + if (!early) {
```

like that.

```
> + stat = kmalloc(sizeof(struct cpu_cgroup_stat)
> + , GFP_KERNEL);
> + for (i = 0; i < CPU_CGROUP_STAT_NSTATS; i++)
> + percpu_counter_init(
> + &stat->cpustat[i], 0);
```

Suppose the kmalloc failed?

```
> + init_task_group.stat = stat;
> + }
> +}
```

more icky layout, and what's that comma doing there?

Again, please use a bit of thought rather than blindly whacking in newlines everywhere.

```
static void cpu_cgroup_initialize(int early)
{
    int i;
    struct cpu_cgroup_stat *stat;

    if (early)
        return;
```

```

stat = kmalloc(sizeof(struct cpu_cgroup_stat), GFP_KERNEL);
for (i = 0; i < CPU_CGROUP_STAT_NSTATS; i++)
    percpu_counter_init(&stat->cpustat[i], 0);
init_task_group.stat = stat;
}

```

is better, yes?

Also, if this code is likely to be executed with any frequency then the test of `early' could be inlined:

```

static inline void cpu_cgroup_initialize(int early)
{
    if (unlikely(!early))
        __cpu_cgroup_initialize();
}

```

yes?

```

> static struct cgroup_subsys_state *
> cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
> {
>     struct task_group *tg, *parent;
>     + int i;
>
>     if (!cgrp->parent) {
>         /* This is early initialization for the top cgroup */
>         @@ -8567,6 +8656,10 @@ cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
>         if (IS_ERR(tg))
>             return ERR_PTR(-ENOMEM);
>
>         + tg->stat = kmalloc(sizeof(struct cpu_cgroup_stat), GFP_KERNEL);
>         + for (i = 0; i < CPU_CGROUP_STAT_NSTATS; i++)
>         +     percpu_counter_init(&tg->stat->cpustat[i], 0);

```

Which will crash the machine if the kmalloc failed.

c'mon guys, that wasn't a great effort.

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