Subject: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Balaji Rao on Wed, 26 Mar 2008 18:18:45 GMT
View Forum Message <> Reply to Message

This patch implements trivial statistics for the cpu controller.

Signed-off-by: Balaji Rao <balajirrao@gmail.com>
CC: Balbir Singh <balbir@linux.vnet.ibm.com>
CC: Dhaval Giani <dhaval@linux.vnet.ibm.com>

```
diff --git a/kernel/sched.c b/kernel/sched.c
index 9fbfa05..eac9333 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -164,10 +164,38 @@ struct cfs_rq;

 static LIST_HEAD(task_groups);

+#ifdef CONFIG_CGROUP_SCHED
+enum cpu_cgroup_stat_index {
+ CPU_CGROUP_STAT_UTIME, /* Usertime of the task group */
+ CPU_CGROUP_STAT_STIME, /* Kerneltime of the task group */
+
+ CPU_CGROUP_STAT_NSTATS,
+};
+
+struct cpu_cgroup_stat_cpu {
+ s64 count[CPU_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct cpu_cgroup_stat {
+ struct cpu_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+/* Called under irq disable. */
+static void __cpu_cgroup_stat_add_safe(struct cpu_cgroup_stat *stat,
+  enum cpu_cgroup_stat_index idx, int val)
+{
+ int cpu = smp_processor_id();
+
+ BUG_ON(!irqs_disabled());
+ stat->cpustat[cpu].count[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
@@ -3670,6 +3698,16 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_UTIME,
+   cputime_to_msecs(cputime));
+ }
+#endif
 }

 /*
@@ -3733,6 +3771,15 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_STIME,
+   cputime_to_msecs(cputime));
+ }
+#endif
 }

 /*
@@ -7939,6 +7986,40 @@ 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;
+ for_each_possible_cpu(cpu)
+  ret += stat->cpustat[cpu].count[idx];
+ return ret;
+}
+
+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
@@ -7961,6 +8042,11 @@ static struct cftype cpu_files[] = {
   .read_u64 = cpu_shares_read_u64,
   .write_u64 = cpu_shares_write_u64,
  },
+
+ {
+  .name = "stat",
+  .read_map = cpu_cgroup_stats_show,
+ },
 #endif
 #ifdef CONFIG_RT_GROUP_SCHED
  {
```

Subject: Re: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Paul Menage on Wed, 26 Mar 2008 19:00:45 GMT
View Forum Message <> Reply to Message

On Wed, Mar 26, 2008 at 11:18 AM, Balaji Rao <balajirrao@gmail.com> wrote:
> +
> +static s64 cpu_cgroup_read_stat(struct cpu_cgroup_stat *stat,
> + enum cpu_cgroup_stat_index idx)
> +{
> + int cpu;
> + s64 ret = 0;
> + for_each_possible_cpu(cpu)
> + ret += stat->cpustat[cpu].count[idx];

On a 32-bit architecture I think this could race with a non-atomic
update that crosses a 32-bit boundary and get a corrupted result.

Paul

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

Subject: Re: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Peter Zijlstra on Wed, 26 Mar 2008 19:58:10 GMT
View Forum Message <> Reply to Message

On Wed, 2008-03-26 at 23:48 +0530, Balaji Rao wrote:
> This patch implements trivial statistics for the cpu controller.
>
> Signed-off-by: Balaji Rao <balajirrao@gmail.com>
> CC: Balbir Singh <balbir@linux.vnet.ibm.com>
> CC: Dhaval Giani <dhaval@linux.vnet.ibm.com>
>
> diff --git a/kernel/sched.c b/kernel/sched.c
> index 9fbfa05..eac9333 100644
> --- a/kernel/sched.c
> +++ b/kernel/sched.c
> @@ -164,10 +164,38 @@ struct cfs_rq;
>
> static LIST_HEAD(task_groups);
>
> +#ifdef CONFIG_CGROUP_SCHED

```
> +enum cpu_cgroup_stat_index {
> + CPU_CGROUP_STAT_UTIME, /* Usertime of the task group */
> + CPU_CGROUP_STAT_STIME, /* Kerneltime of the task group */
> +
> + CPU_CGROUP_STAT_NSTATS,
> +};
> +
> +struct cpu_cgroup_stat_cpu {
> + s64 count[CPU_CGROUP_STAT_NSTATS];
> +} ____cacheline_aligned_in_smp;
> +
> +struct cpu_cgroup_stat {
> + struct cpu_cgroup_stat_cpu cpustat[NR_CPUS];
> +};
> +
> +/* Called under irq disable. */
> +static void __cpu_cgroup_stat_add_safe(struct cpu_cgroup_stat *stat,
> +  enum cpu_cgroup_stat_index idx, int val)
```

What is safe about this function?

```
> +{
> + int cpu = smp_processor_id();
> +
> + BUG_ON(!irqs_disabled());
> + stat->cpustat[cpu].count[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
> @@ -3670,6 +3698,16 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_UTIME,
```

> + cputime_to_msecs(cputime));
> + }
> +#endif
> }
>
> /*
> @@ -3733,6 +3771,15 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_STIME,
> + cputime_to_msecs(cputime));
> + }
> +#endif
> }

So both of these are tick based? The normal CFS [us]time stats are not.

> /*
> @@ -7939,6 +7986,40 @@ 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;
> + for_each_possible_cpu(cpu)
> + ret += stat->cpustat[cpu].count[idx];
> + return ret;
> +}
> +
> +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
> @@ -7961,6 +8042,11 @@ static struct cftype cpu_files[] = {
>     .read_u64 = cpu_shares_read_u64,
>     .write_u64 = cpu_shares_write_u64,
>   },
> +
> + {
> +  .name = "stat",
> +  .read_map = cpu_cgroup_stats_show,
> + },
>  #endif
>  #ifdef CONFIG_RT_GROUP_SCHED
>   {
>
> --
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at  http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at  http://www.tux.org/lkml/
```



Subject: Re: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Balaji Rao on Fri, 28 Mar 2008 10:02:37 GMT
View Forum Message <> Reply to Message

On Thursday 27 March 2008 01:28:10 am Peter Zijlstra wrote:

> On Wed, 2008-03-26 at 23:48 +0530, Balaji Rao wrote:
<snip>
> > +/* Called under irq disable. */
> > +static void __cpu_cgroup_stat_add_safe(struct cpu_cgroup_stat *stat,
> > +  enum cpu_cgroup_stat_index idx, int val)
>
> What is safe about this function?
>
That it can be called only from an interrupt context.
> > +{
> > + int cpu = smp_processor_id();
> > +
> > + BUG_ON(!irqs_disabled());
> > + stat->cpustat[cpu].count[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
> > @@ -3670,6 +3698,16 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_UTIME,
> > +   cputime_to_msecs(cputime));
> > + }
> > +#endif
> >  }
> >
> >  /*
> > @@ -3733,6 +3771,15 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_STIME,
> > + cputime_to_msecs(cputime));
> > + }
> > +#endif
> > }
>
> So both of these are tick based? The normal CFS [us]time stats are not.
>
Hmmm.. Yea, right. So I should use the approach used by task_utime and task_stime when reporting it, right ?
> > /*
> > @@ -7939,6 +7986,40 @@ static u64 cpu_shares_read_u64(struct cgroup *cgrp, struct cftype *cft)
> >
> > return (u64) tg->shares;
> > }
> > +

Thanks for the review.
--
regards,
balaji rao

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

---

Subject: Re: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Peter Zijlstra on Fri, 28 Mar 2008 10:17:16 GMT
View Forum Message <> Reply to Message

On Fri, 2008-03-28 at 15:32 +0530, Balaji Rao wrote:
> On Thursday 27 March 2008 01:28:10 am Peter Zijlstra wrote:
> > On Wed, 2008-03-26 at 23:48 +0530, Balaji Rao wrote:
> <snip>
> > > +/* Called under irq disable. */
> > > +static void __cpu_cgroup_stat_add_safe(struct cpu_cgroup_stat *stat,
> > > + enum cpu_cgroup_stat_index idx, int val)
> >
> > What is safe about this function?
> >
> That it can be called only from an interrupt context.

It can be called from any context that has hardirqs disabled. And the __
prefix suggests as much, no need to tag _safe to the end as well, we
never do that.

> > > +{
> > > + int cpu = smp_processor_id();
> > > +
> > > + BUG_ON(!irqs_disabled());
> > > + stat->cpustat[cpu].count[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
> > > @@ -3670,6 +3698,16 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_UTIME,
> > > +   cputime_to_msecs(cputime));
> > > + }
> > > +#endif
> > >  }
> > >
> > >  /*
> > > @@ -3733,6 +3771,15 @@ 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_safe(&tg->stat, CPU_CGROUP_STAT_STIME,
> > > +   cputime_to_msecs(cputime));
> > > + }
> > > +#endif
> > >  }
> >
> > So both of these are tick based? The normal CFS [us]time stats are not.
> >
> Hmmm.. Yea, right. So I should use the approach used by task_utime and task_stime when reporting it, right ?

Not sure what you want to use this for, but yeah, that makes most sense.

That is, I do know _what_ you want to use it for, just not sure which
requirements you put on it. The pure tick based thing might be good
enough for most purposes, the runtime proportion thing is just more
accurate.

> > >  /*
> > > @@ -7939,6 +7986,40 @@ static u64 cpu_shares_read_u64(struct cgroup *cgrp, struct cftype *cft)
> > >
> > >   return (u64) tg->shares;
> > >  }
> > > +
>
> Thanks for the review.