Subject: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Balaji Rao on Sat, 05 Apr 2008 18:09:46 GMT
View Forum Message <> Reply to Message

This patch implements some trivial statistics for the CPU controller.

As per our current requirements, we have decided to keep the stats tick based as opposed to using CFS precise accounting.

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 8206fda..e2acf06 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(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
@@ -3733,6 +3761,16 @@ 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
 }

 /*
@@ -3796,6 +3834,15 @@ 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
 }

 /*
@@ -8004,6 +8051,45 @@ 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;
+
+ local_irq_save(flags);
+ for_each_possible_cpu(cpu)
+  ret += stat->cpustat[cpu].count[idx];
+ local_irq_restore(flags);
+
+ 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
@@ -8026,6 +8112,11 @@ 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
  {

--
regards,
Balaji Rao
Dept. of Mechanical Engineering,
National Institute of Technology Karnataka, India

_____
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 Dhaval Giani on Sat, 05 Apr 2008 18:56:16 GMT
View Forum Message <> Reply to Message

On Sat, Apr 05, 2008 at 11:39:46PM +0530, Balaji Rao wrote:
> This patch implements some trivial statistics for the CPU controller.
>
> As per our current requirements, we have decided to keep the stats tick based as opposed to using CFS precise accounting.
>

A quick question? Why? Everywhere in the CFS we use precise accounting,
(even within the CPU controller). Doesn't make sense using ticks here
then.

Will review and respond soon.

Thanks,
Dhaval

> 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 8206fda..e2acf06 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(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
> @@ -3733,6 +3761,16 @@ 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
>  }
>
>  /*
> @@ -3796,6 +3834,15 @@ 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
>  }
>
>  /*
> @@ -8004,6 +8051,45 @@ 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;
> +
> + local_irq_save(flags);
> + for_each_possible_cpu(cpu)
> +  ret += stat->cpustat[cpu].count[idx];
> + local_irq_restore(flags);
> +
> + 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
> @@ -8026,6 +8112,11 @@ 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
>   {
>
> --
> regards,
> Balaji Rao
> Dept. of Mechanical Engineering,
> National Institute of Technology Karnataka, India

--
regards,
Dhaval
_____
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 Balaji Rao on Sat, 05 Apr 2008 19:09:02 GMT
View Forum Message <> Reply to Message

On Sunday 06 April 2008 12:26:16 am Dhaval Giani wrote:
> On Sat, Apr 05, 2008 at 11:39:46PM +0530, Balaji Rao wrote:
> > This patch implements some trivial statistics for the CPU controller.
> >
> > As per our current requirements, we have decided to keep the stats tick based as opposed to using CFS precise accounting.
> >
>
> A quick question? Why? Everywhere in the CFS we use precise accounting,
> (even within the CPU controller). Doesn't make sense using ticks here
> then.
>
Using CFS precise accounting has a cost (see task_utime and task_stime). Currently our requirements don't call for precise statistics.

--
regards,
Balaji Rao
Dept. of Mechanical Engineering,
National Institute of Technology Karnataka, India

_____
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 Dhaval Giani on Sat, 05 Apr 2008 19:40:41 GMT
View Forum Message <> Reply to Message

On Sat, Apr 05, 2008 at 11:39:46PM +0530, Balaji Rao wrote:
> This patch implements some trivial statistics for the CPU controller.
>
> As per our current requirements, we have decided to keep the stats tick based as opposed to using CFS precise accounting.
>
> 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 8206fda..e2acf06 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,

why the extra space?

> +};
> +
> +struct cpu_cgroup_stat_cpu {
> + s64 count[CPU_CGROUP_STAT_NSTATS];

u64? time does not go negative :)
count also is not very clear? Can you give a more descriptive name?

> +} ____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(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
> @@ -3733,6 +3761,16 @@ 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
> }
>
> /*
> @@ -3796,6 +3834,15 @@ 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
> }
>
> /*
> @@ -8004,6 +8051,45 @@ 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;
> +
> + local_irq_save(flags);

I am just wondering. Is local_irq_save() enough?

> + for_each_possible_cpu(cpu)
> + ret += stat->cpustat[cpu].count[idx];
> + local_irq_restore(flags);
> +
> + 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
> @@ -8026,6 +8112,11 @@ 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
>   {
>

--

regards,
Dhaval

_____
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 Balaji Rao on Sat, 05 Apr 2008 20:39:57 GMT
View Forum Message <> Reply to Message

On Sunday 06 April 2008 01:10:41 am Dhaval Giani wrote:
<snip>
> >
> > +#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,
>
> why the extra space?
Just to keep things clearly separated. If you've not noticed,
CPU_CGROUP_STAT_NSTATS is not a stat.
>
> > +};
> > +
> > +struct cpu_cgroup_stat_cpu {
> > + s64 count[CPU_CGROUP_STAT_NSTATS];
>
> u64? time does not go negative :)
Right. But these stats are not only going to measure time. We need the same
variables for measuring other stats as well. I'm not sure if we would
encounter scheduler stats that would count negative.

Balbir, what do you say ?

> count also is not very clear? Can you give a more descriptive name?
>
ok. How does 'value' look ?

<snip>

> > +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;
>
> > +
> > + local_irq_save(flags);
>
> I am just wondering. Is local_irq_save() enough?
>
Hmmm.. You are right.This does not prevent concurrent updates on other CPUs
from crossing a 32bit boundary. Am not sure how to do this in a safe way. I
can only think of using atomic64_t now..

> > + for_each_possible_cpu(cpu)
> > + ret += stat->cpustat[cpu].count[idx];
> > + local_irq_restore(flags);
> > +
> > + return ret;
> > +}
> > +

Thanks for the review.

--
regards,
Balaji Rao
Dept. of Mechanical Engineering,
National Institute of Technology Karnataka, India

Subject: Re: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Dhaval Giani on Sat, 05 Apr 2008 21:01:13 GMT
View Forum Message <> Reply to Message

On Sun, Apr 06, 2008 at 02:01:52AM +0530, Balaji Rao wrote:
> On Sunday 06 April 2008 01:10:41 am Dhaval Giani wrote:
> > > > +};
> > > > +
> > > > +struct cpu_cgroup_stat_cpu {
> > > > + s64 count[CPU_CGROUP_STAT_NSTATS];
> > >
> > > u64? time does not go negative :)
> > Right. But these stats are not only going to measure time. We need the same
> > variables for measuring other stats as well. I'm not sure if we would
> > encounter scheduler stats that would count negative.

>
> Balbir, what do you say ?

I would prefer to keep the stats logically separate. So something like
struct cpu_cgroup_stat_cpu {
 u64 time[];
 s64 some_other_stat;
}
and so on. (I am not sure, is there some advantage gained by using
structs?) Makes the code more maintainable imho.

>
> > count also is not very clear? Can you give a more descriptive name?
> >
> ok. How does 'value' look ?
>
> <snip>
>
> > > +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;
> > >
> > > +
> > > + local_irq_save(flags);
> > >
> > I am just wondering. Is local_irq_save() enough?
> >
> Hmmm.. You are right.This does not prevent concurrent updates on other CPUs
> from crossing a 32bit boundary. Am not sure how to do this in a safe way. I
> can only think of using atomic64_t now..
>

I am going to answer that one when I am awake :-)

--
regards,
Dhaval

_____
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 Balaji Rao on Sat, 05 Apr 2008 21:31:05 GMT
View Forum Message <> Reply to Message

On Sunday 06 April 2008 02:29:14 am Dhaval Giani wrote:
> On Sun, Apr 06, 2008 at 02:01:52AM +0530, Balaji Rao wrote:
> > On Sunday 06 April 2008 01:10:41 am Dhaval Giani wrote:
> > > > > > +};
> > > > > +
> > > > > +struct cpu_cgroup_stat_cpu {
> > > > > + s64 count[CPU_CGROUP_STAT_NSTATS];
> > > >
> > > u64? time does not go negative :)
> > Right. But these stats are not only going to measure time. We need the
same
> > variables for measuring other stats as well. I'm not sure if we would
> > encounter scheduler stats that would count negative.
> >
> > Balbir, what do you say ?
>
> I would prefer to keep the stats logically separate. So something like
> struct cpu_cgroup_stat_cpu {
> u64 time[];
> s64 some_other_stat;
> }
> and so on. (I am not sure, is there some advantage gained by using
> structs?) Makes the code more maintainable imho.
>
This would break the generic nature of __cpu_cgroup_stat_add. Its not a nice
thing in my opinion.

--
regards,
Balaji Rao
Dept. of Mechanical Engineering,
National Institute of Technology Karnataka, India

_____
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 Balbir Singh on Sun, 06 Apr 2008 05:12:17 GMT
View Forum Message <> Reply to Message

Balaji Rao wrote:
> On Sunday 06 April 2008 02:29:14 am Dhaval Giani wrote:
>> On Sun, Apr 06, 2008 at 02:01:52AM +0530, Balaji Rao wrote:

>>> On Sunday 06 April 2008 01:10:41 am Dhaval Giani wrote:
>>>>> +};
>>>>> +
>>>>> +struct cpu_cgroup_stat_cpu {
>>>>> + s64 count[CPU_CGROUP_STAT_NSTATS];
>>>> u64? time does not go negative :)
>>> Right. But these stats are not only going to measure time. We need the
> same
>>> variables for measuring other stats as well. I'm not sure if we would
>>> encounter scheduler stats that would count negative.
>>>
>>> Balbir, what do you say ?
>> I would prefer to keep the stats logically separate. So something like
>> struct cpu_cgroup_stat_cpu {
>> u64 time[];
>> s64 some_other_stat;
>> }
>> and so on. (I am not sure, is there some advantage gained by using
>> structs?) Makes the code more maintainable imho.
>>
> This would break the generic nature of __cpu_cgroup_stat_add. Its not a nice
> thing in my opinion.
>

I prefer keeping stats in the array as Balaji has done, it makes it easier to do
batch processing on the stats.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____
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 Mon, 07 Apr 2008 13:24:53 GMT
View Forum Message <> Reply to Message

On Sun, 2008-04-06 at 02:01 +0530, Balaji Rao wrote:

> > > +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;
> >
> > > +
> > > + local_irq_save(flags);
> >
> > I am just wondering. Is local_irq_save() enough?
> >
> Hmmm.. You are right.This does not prevent concurrent updates on other CPUs
> from crossing a 32bit boundary. Am not sure how to do this in a safe way. I
> can only think of using atomic64_t now..
>
> > > + for_each_possible_cpu(cpu)
> > > + ret += stat->cpustat[cpu].count[idx];
> > > + local_irq_restore(flags);
> > > +
> > > + return ret;
> > > +}
> > > +

So many stats to steal code from,.. but you didn't :-(

Look at mm/vmstat.c, that is a rather complete example.

The trick to solving the above is to use per cpu deltas instead, the
deltas can be machine word size and are thus always read in an atomic
manner (provided they are also naturally aligned).

_____
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 Balaji Rao on Thu, 10 Apr 2008 16:09:59 GMT
View Forum Message <> Reply to Message

On Monday 07 April 2008 06:54:53 pm Peter Zijlstra wrote:
> On Sun, 2008-04-06 at 02:01 +0530, Balaji Rao wrote:
>
> > > > > +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;
> > >
> > > > +
> > > > + local_irq_save(flags);
> > >
> > > I am just wondering. Is local_irq_save() enough?
> > >
> > Hmmm.. You are right.This does not prevent concurrent updates on other
CPUs
> > from crossing a 32bit boundary. Am not sure how to do this in a safe way.
I
> > can only think of using atomic64_t now..
> >
> > > > > + for_each_possible_cpu(cpu)
> > > > > + ret += stat->cpustat[cpu].count[idx];
> > > > > + local_irq_restore(flags);
> > > > > +
> > > > > + return ret;
> > > > > +}
> > > > > +
>
> So many stats to steal code from,.. but you didn't :-(
>
> Look at mm/vmstat.c, that is a rather complete example.
>
> The trick to solving the above is to use per cpu deltas instead, the
> deltas can be machine word size and are thus always read in an atomic
> manner (provided they are also naturally aligned).
>
>
Hi Peter,

This wont work for time based statistics. At nsec granularity, a word can hold
a time value of up to ~4s.

I propose to solve this problem by using a lock to protect the statistics, but
only on 32bit architectures.

I'm not sure how good a solution this is, but that's the best I can think of
ATM.
--
regards,
Balaji Rao
Dept. of Mechanical Engineering,
National Institute of Technology Karnataka, India

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [RFC][-mm] [1/2] Simple stats for cpu resource controller
Posted by Peter Zijlstra on Thu, 10 Apr 2008 16:25:11 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-10 at 21:39 +0530, Balaji Rao wrote:
> On Monday 07 April 2008 06:54:53 pm Peter Zijlstra wrote:
> > On Sun, 2008-04-06 at 02:01 +0530, Balaji Rao wrote:
> >
> > > > > > > +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;
> > > > > >
> > > > > > > +
> > > > > > > + local_irq_save(flags);
> > > > > >
> > > > > I am just wondering. Is local_irq_save() enough?
> > > > >
> > > > Hmmm.. You are right.This does not prevent concurrent updates on other
> > CPUs
> > > > from crossing a 32bit boundary. Am not sure how to do this in a safe way.
> > I
> > > > can only think of using atomic64_t now..
> > > >
> > > > > > > + for_each_possible_cpu(cpu)
> > > > > > > + ret += stat->cpustat[cpu].count[idx];
> > > > > > > + local_irq_restore(flags);
> > > > > > > +
> > > > > > > + return ret;
> > > > > > > +}
> > > > > > > +
> > >
> > So many stats to steal code from,.. but you didn't :-(
> >
> > Look at mm/vmstat.c, that is a rather complete example.
> >
> > The trick to solving the above is to use per cpu deltas instead, the
> > deltas can be machine word size and are thus always read in an atomic
> > manner (provided they are also naturally aligned).
> >
> >
> Hi Peter,
>

> This wont work for time based statistics. At nsec granularity, a word can hold
> a time value of up to ~4s.

4 seconds is plenty for a delta, most increments are in the ms range.

> I propose to solve this problem by using a lock to protect the statistics, but
> only on 32bit architectures.
>
> I'm not sure how good a solution this is, but that's the best I can think of
> ATM.

Not needed, keep per cpu word deltas and fold into a global u64 counter
while holding a lock.

Subject: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by Balaji Rao on Thu, 01 May 2008 17:41:06 GMT
View Forum Message <> Reply to Message

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 <balajirrao@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

```
+
+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];
+ u32 delta[CPU_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct cpu_cgroup_stat {
+ struct cpu_cgroup_stat_cpu cpustat[NR_CPUS];
+ spinlock_t lock;
+};
+
+/* 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());
+ stat->cpustat[cpu].delta[idx] += val;
+
+ if (stat->cpustat[cpu].delta[idx] > CPU_CGROUP_STAT_THRESHOLD) {
+  spin_lock_irqsave(&stat->lock, flags);
+  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;
+}
+
+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,
+ },
 };

 static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
```

--
Warm Regards,

Balaji Rao
Dept. of Mechanical Engineering
NITK

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

---

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 <balajirrao@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 <balajirrao@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, /* 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];
> + 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.

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by Balaji Rao on Fri, 02 May 2008 19:40:28 GMT
View Forum Message <> Reply to Message

On Friday 02 May 2008 02:30:26 am Andrew Morton wrote:
<snip>

Hi Andrew,

Thank you for the review.
>
> 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.
No, I hadn't consider using the percpu_counters infrastructure. But today when I tried using it, I got an early exception.I guess its because I tried calling percpu_counter_init from within sched_init, which I perhaps shouldn't do, because percpu_counter_init expects cpu hotplug code to be initialized by then. Right ? Correct me if I'm wrong.

How about we start collecting statistics at a later stage i.e, after percpu_counter becomes usable ?

--
Warm Regards,

Balaji Rao
Dept. of Mechanical Engineering
NITK

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

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by akpm on Fri, 02 May 2008 19:53:04 GMT
View Forum Message <> Reply to Message

On Sat, 3 May 2008 01:10:28 +0530
Balaji Rao <balajirrao@gmail.com> wrote:

> On Friday 02 May 2008 02:30:26 am Andrew Morton wrote:
> <snip>
>

> Hi Andrew,
>
> Thank you for the review.
> >
> > 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.
> No, I hadn't consider using the percpu_counters infrastructure. But today when
> I tried using it, I got an early exception.I guess its because I tried
> calling percpu_counter_init from within sched_init, which I perhaps shouldn't
> do, because percpu_counter_init expects cpu hotplug code to be initialized by
> then. Right ? Correct me if I'm wrong.

I don't see any reason why we cannot run percpu_counter_init() prior to
running percpu_counter_startup(). And it is desirable that we be able to
start using the percpu-counters quite early.

Can you debug it a bit please? It's probably some silly little thing,
perhaps fixable by calling percpu_counter_startup() earlier.

> How about we start collecting statistics at a later stage i.e, after
> percpu_counter becomes usable ?

It would be better to make the core infrastructure more robust, rather
than working around problems it might have.

It's rather nice that percpu_counters internally take care of
cpu-hotplugging, and use cpu_online_map. I was amazed at how easily that
was added. I still expect it to break somehow..

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

---

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by Balaji Rao on Fri, 02 May 2008 22:47:03 GMT
View Forum Message <> Reply to Message

On Saturday 03 May 2008 01:23:04 am Andrew Morton wrote:
> On Sat, 3 May 2008 01:10:28 +0530
>
> Balaji Rao <balajirrao@gmail.com> wrote:
> > On Friday 02 May 2008 02:30:26 am Andrew Morton wrote:

> > <snip>
> >
> > Hi Andrew,
> >
> > Thank you for the review.
> >
> > > 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.
> >
> > No, I hadn't consider using the percpu_counters infrastructure. But today
> > when I tried using it, I got an early exception.I guess its because I
> > tried calling percpu_counter_init from within sched_init, which I perhaps
> > shouldn't do, because percpu_counter_init expects cpu hotplug code to be
> > initialized by then. Right ? Correct me if I'm wrong.
>
> I don't see any reason why we cannot run percpu_counter_init() prior to
> running percpu_counter_startup(). And it is desirable that we be able to
> start using the percpu-counters quite early.
>
> Can you debug it a bit please? It's probably some silly little thing,
> perhaps fixable by calling percpu_counter_startup() earlier.
>
percpu_counter_init uses kmalloc to create percpu counters. This raises an
early exception as kmem_cache is not initialized that early.

It worked for me if we statically allocate memory for the counters. But its
not at all a nice thing to do and I don't see another way to make it fit for
early use.

I'm beginning to run out of ideas! Why not do what I earlier suggested - begin
collecting statistics once we are able to safely use percpu_counters ? This
now seems to be the best alternative IMHO.

--
Warm Regards,

Balaji Rao
Dept. of Mechanical Engineering
NITK

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by akpm on Fri, 02 May 2008 23:04:34 GMT
View Forum Message <> Reply to Message

On Sat, 3 May 2008 04:17:03 +0530
Balaji Rao <balajirrao@gmail.com> wrote:

> On Saturday 03 May 2008 01:23:04 am Andrew Morton wrote:
> > On Sat, 3 May 2008 01:10:28 +0530
> >
> > Balaji Rao <balajirrao@gmail.com> wrote:
> > > On Friday 02 May 2008 02:30:26 am Andrew Morton wrote:
> > > <snip>
> > >
> > > Hi Andrew,
> > >
> > > Thank you for the review.
> > >
> > > > 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.
> > >
> > > No, I hadn't consider using the percpu_counters infrastructure. But today
> > > when I tried using it, I got an early exception.I guess its because I
> > > tried calling percpu_counter_init from within sched_init, which I perhaps
> > > shouldn't do, because percpu_counter_init expects cpu hotplug code to be
> > > initialized by then. Right ? Correct me if I'm wrong.
> >
> > I don't see any reason why we cannot run percpu_counter_init() prior to
> > running percpu_counter_startup(). And it is desirable that we be able to
> > start using the percpu-counters quite early.
> >
> > Can you debug it a bit please? It's probably some silly little thing,
> > perhaps fixable by calling percpu_counter_startup() earlier.
> >
> percpu_counter_init uses kmalloc to create percpu counters. This raises an
> early exception as kmem_cache is not initialized that early.

whaa? kmalloc is ready to be used quite early in boot. It's a bit of a
concern that the CPU resource controller is doing stuff before even kmalloc
is ready to go.

What's the call path here? Via cgroup_init_early()? Does it need to run
that early?

> It worked for me if we statically allocate memory for the counters. But its

> not at all a nice thing to do and I don't see another way to make it fit for
> early use.
>
> I'm beginning to run out of ideas! Why not do what I earlier suggested - begin
> collecting statistics once we are able to safely use percpu_counters ? This
> now seems to be the best alternative IMHO.

I'd need to see the code. If we end up doing

```
 if (counters_are_ready)
  increment_counter();
```

all over then place then we need to think harder.

Maybe we need a cgroup_init_late(), which can do memory allocations. If
nothing actually needs to touch the counters before cgroup_init_late() runs
then that might be OK.

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3
Posted by Balaji Rao on Fri, 02 May 2008 23:56:46 GMT
View Forum Message <> Reply to Message

On Saturday 03 May 2008 05:11:33 am Andrew Morton wrote:

<snip>
> > > >
> > > > percpu_counter_init uses kmalloc to create percpu counters. This
> > > > raises an early exception as kmem_cache is not initialized that
> > > > early.
> > > >
> > > whaa? kmalloc is ready to be used quite early in boot. It's a bit of
> > > a concern that the CPU resource controller is doing stuff before even
> > > kmalloc is ready to go.
> > >
> > No it doesn't. I placed the call to percpu_counter_init in sched_init.
> > Hence the problem.
>
> I'm groping in the dark without seeing the diff.
oh! am sorry about that! some cosmetic changes remain. It also contains some
modifications done to the percpu_counter core.

<snip>

> > For the group_init_late() which you suggested, we should probably hook
> > into a place where kmalloc is ready and
> > account_user_time/account_system_time is not yet called even once. Does
> > it make sense ?
>
> yes, that would be good.
OK, so when does account_system_time get called for the first time ? after
IRQs are set up, is it ? So, where do we place the hook ?

Here's the patch.

```
diff --git a/include/linux/percpu_counter.h b/include/linux/percpu_counter.h
index 9007ccd..8a1b756 100644
--- a/include/linux/percpu_counter.h
+++ b/include/linux/percpu_counter.h
@@ -21,7 +21,7 @@ struct percpu_counter {
 #ifdef CONFIG_HOTPLUG_CPU
   struct list_head list; /* All percpu_counters are on a list */
 #endif
- s32 *counters;
+ s32 counters[NR_CPUS];
 };

 #if NR_CPUS >= 16
diff --git a/kernel/sched.c b/kernel/sched.c
index bbdc32a..fde11f8 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -68,6 +68,7 @@
 #include <linux/hrtimer.h>
 #include <linux/tick.h>
 #include <linux/ftrace.h>
+#include <linux/percpu_counter.h>

 #include <asm/tlb.h>
 #include <asm/irq_regs.h>
@@ -248,10 +249,34 @@ 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, /* Usertime of the task group */
+ CPU_CGROUP_STAT_STIME, /* Kerneltime of the task group */
+
```

```
+ CPU_CGROUP_STAT_NSTATS,
+};
+
+struct cpu_cgroup_stat {
+ struct percpu_counter cpustat[CPU_CGROUP_STAT_NSTATS];
+};
+
+/* Called under irq disable. */
+static void __cpu_cgroup_stat_add(struct cpu_cgroup_stat *stat,
+  enum cpu_cgroup_stat_index idx, int val)
+{
+  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
@@ -3837,6 +3862,16 @@ 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 +3935,15 @@ 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
 }

 /*
@@ -7426,6 +7470,10 @@ void __init sched_init(void)
 #ifdef CONFIG_GROUP_SCHED
  list_add(&init_task_group.list, &task_groups);
 #endif
+
+ for(i = 0; i < CPU_CGROUP_STAT_NSTATS; i++) {
+  percpu_counter_init(&init_task_group.stat.cpustat[i], 0);
+ }

  for_each_possible_cpu(i) {
   struct rq *rq;
@@ -8183,10 +8231,12 @@ static struct cgroup_subsys_state *
 cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
 {
  struct task_group *tg;
+ int i;

  if (!cgrp->parent) {
   /* This is early initialization for the top cgroup */
   init_task_group.css.cgroup = cgrp;
+  /* Initialize init_task_grp's stat object */
   return &init_task_group.css;
  }

@@ -8201,6 +8251,10 @@ cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
  /* Bind the cgroup to task_group object we just created */
  tg->css.cgroup = cgrp;

+ /* Initialize the stats object */
+ for(i = 0; i < CPU_CGROUP_STAT_NSTATS; i++)
+  percpu_counter_init(&tg->stat.cpustat[i], 0);
+
  return &tg->css;
 }

@@ -8249,6 +8303,36 @@ 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)
+{
+ return 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, },
+};
+
+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++) {
+  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 +8379,10 @@ static struct cftype cpu_files[] = {
   .write_u64 = cpu_rt_period_write_uint,
  },
 #endif
+ {
+  .name = "stat",
+  .read_map = cpu_cgroup_stats_show,
+ },
 };

 static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
@@ -8310,7 +8398,7 @@ struct cgroup_subsys cpu_cgroup_subsys = {
  .attach  = cpu_cgroup_attach,
  .populate = cpu_cgroup_populate,
  .subsys_id = cpu_cgroup_subsys_id,
```

```
- .early_init = 1,
+// .early_init = 1,
 };

 #endif /* CONFIG_CGROUP_SCHED */
diff --git a/lib/percpu_counter.c b/lib/percpu_counter.c
index 1191744..a7e84f9 100644
--- a/lib/percpu_counter.c
+++ b/lib/percpu_counter.c
@@ -14,6 +14,7 @@ static LIST_HEAD(percpu_counters);
 static DEFINE_MUTEX(percpu_counters_lock);
 #endif

+#define per_cpu_ptr(var,cpu) &var[cpu]
 void percpu_counter_set(struct percpu_counter *fbc, s64 amount)
 {
  int cpu;
@@ -74,7 +75,6 @@ int percpu_counter_init(struct percpu_counter *fbc, s64 amount)
 {
  spin_lock_init(&fbc->lock);
  fbc->count = amount;
- fbc->counters = alloc_percpu(s32);
  if (!fbc->counters)
   return -ENOMEM;
 #ifdef CONFIG_HOTPLUG_CPU
@@ -102,7 +102,6 @@ void percpu_counter_destroy(struct percpu_counter *fbc)
   return;

  free_percpu(fbc->counters);
- fbc->counters = NULL;
 #ifdef CONFIG_HOTPLUG_CPU
  mutex_lock(&percpu_counters_lock);
  list_del(&fbc->list);
```

--
Warm Regards,

Balaji Rao
Dept. of Mechanical Engineering
NITK

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v3

On Sat, 3 May 2008 05:26:46 +0530
Balaji Rao <balajirrao@gmail.com> wrote:

> > yes, that would be good.
> OK, so when does account_system_time get called for the first time ? after
> IRQs are set up, is it ? So, where do we place the hook ?

Don't know - I'd need to dive in and work that out, and it's probably
better than you do this..

> Here's the patch.
>
> diff --git a/include/linux/percpu_counter.h b/include/linux/percpu_counter.h
> index 9007ccd..8a1b756 100644
> --- a/include/linux/percpu_counter.h
> +++ b/include/linux/percpu_counter.h
> @@ -21,7 +21,7 @@ struct percpu_counter {
> #ifdef CONFIG_HOTPLUG_CPU
>   struct list_head list; /* All percpu_counters are on a list */
> #endif
> - s32 *counters;
> + s32 counters[NR_CPUS];
> };

Please, no. That's a 4092-byte increase in sizeof(struct percpu_counter).
Hence a 12 kbyte increase in sizeof(struct ext3_sb_info). Let's just sort
out the cgroup startup ordering.

<looks at __percpu_alloc_mask>
<wanders off-topic>

Eric, is that optimal? alloc_percpu() will pass down cpu_possible_map in
`mask', and we only need to allocate enough slots to cover the
highest-set-bit in cpu_possible_map. However the implementation ignores
`mask' and does

```
size_t sz = roundup(nr_cpu_ids * sizeof(void *), cache_line_size());
void *pdata = kzalloc(sz, gfp);
```

Now, if the highest-set-bit in cpu_possible_map is always equal to
(1<<nr_cpu_ids) then it doesn't matter. But is that the case?

(If someone calls __percpu_alloc_mask with something that has less bits set
than cpu_possible_map then it surely is wasteful, but that sounds

unlikely).

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

---

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v4
Posted by Balaji Rao on Sun, 11 May 2008 20:18:37 GMT
View Forum Message <> Reply to Message

Hi Andrew,

Here's a version that uses percpu_counters and which actually works. The
only evil it contains is the check,

```
 if (percpu_counter_ready) {
  ..
 }
```

I've also added a callback called 'initalize' to a cgroup_subsys.

The alternate idea I suggested was to add a hook at a point where the first
task has not yet started and where kmalloc is usable. Its hacky and not a
nice thing to do IMHO.

How does this look ?

Regards,
Balaji Rao

Signed-off-by: Balaji Rao <balajirrao@gmail.com>

```
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 8877a08..be37395 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);
  }

  /* Add init_css_set to the hash table */
diff --git a/kernel/sched.c b/kernel/sched.c
index bbdc32a..50f737e 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -247,11 +247,32 @@ static void destroy_rt_bandwidth(struct rt_bandwidth *rt_b)
 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, /* Usertime of the task group */
+ CPU_CGROUP_STAT_STIME, /* Kerneltime 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, int 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
@@ -3837,6 +3858,16 @@ 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
 }

 /*
@@ -3892,8 +3923,17 @@ 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));
+  }
+ }
+#endif
  else if (atomic_read(&rq->nr_iowait) > 0)
   cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
```

```
  else
@@ -8179,10 +8219,26 @@ static inline struct task_group *cgroup_tg(struct cgroup *cgrp)
        struct task_group, css);
 }

+static void cpu_cgroup_initialize(int early)
+{
+ int i;
+ struct cpu_cgroup_stat *stat;
+
+ if (!early) {
+  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;
+ }
+}
+
 static struct cgroup_subsys_state *
 cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
 {
  struct task_group *tg;
+ int i;

  if (!cgrp->parent) {
   /* This is early initialization for the top cgroup */
@@ -8198,6 +8254,10 @@ 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);
+
  /* Bind the cgroup to task_group object we just created */
  tg->css.cgroup = cgrp;

@@ -8251,6 +8311,38 @@ static u64 cpu_shares_read_u64(struct cgroup *cgrp, struct cftype
*cft)
 }
 #endif

+static s64 cpu_cgroup_read_stat(struct cpu_cgroup_stat *stat,
+  enum cpu_cgroup_stat_index idx)
+{
+ if (stat)
```

```
+  return percpu_counter_read(&stat->cpustat[idx]);
+
+ return 0;
+}
+
+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 < CPU_CGROUP_STAT_NSTATS; 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;
+}
+
 #ifdef CONFIG_RT_GROUP_SCHED
 static ssize_t cpu_rt_runtime_write(struct cgroup *cgrp, struct cftype *cft,
     s64 val)
@@ -8295,6 +8387,10 @@ static struct cftype cpu_files[] = {
   .write_u64 = cpu_rt_period_write_uint,
  },
 #endif
+ {
+  .name = "stat",
+  .read_map = cpu_cgroup_stats_show,
+ },
 };

 static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
@@ -8304,6 +8400,7 @@ static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct
cgroup *cont)

 struct cgroup_subsys cpu_cgroup_subsys = {
  .name  = "cpu",
+ .initialize = cpu_cgroup_initialize,
```

```
  .create  = cpu_cgroup_create,
  .destroy = cpu_cgroup_destroy,
  .can_attach = cpu_cgroup_can_attach,
```


Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v4
Posted by Li Zefan on Mon, 12 May 2008 02:54:45 GMT
View Forum Message <> Reply to Message

Balaji Rao wrote:
> /*
> @@ -3892,8 +3923,17 @@ 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));
> + }
> + }

You should put this '}' after '#endif'

> +#endif
> else if (atomic_read(&rq->nr_iowait) > 0)
> cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
> else
> @@ -8179,10 +8219,26 @@ static inline struct task_group *cgroup_tg(struct cgroup *cgrp)
> struct task_group, css);
> }
>
> +static void cpu_cgroup_initialize(int early)
> +{
> + int i;
> + struct cpu_cgroup_stat *stat;
> +
> + if (!early) {
> + 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;
> + }
> +}
> +
> static struct cgroup_subsys_state *
> cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
> {
>   struct task_group *tg;
> + int i;
>
>   if (!cgrp->parent) {
>   /* This is early initialization for the top cgroup */
> @@ -8198,6 +8254,10 @@ 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);
> +

I guess you forgot to free those things in cpu_cgroup_destroy().

>   /* Bind the cgroup to task_group object we just created */
>   tg->css.cgroup = cgrp;
>
> @@ -8251,6 +8311,38 @@ static u64 cpu_shares_read_u64(struct cgroup *cgrp, struct cftype
*cft)
> }
> #endif
>
> +static s64 cpu_cgroup_read_stat(struct cpu_cgroup_stat *stat,
> + enum cpu_cgroup_stat_index idx)
> +{
> + if (stat)
> + return percpu_counter_read(&stat->cpustat[idx]);
> +
> + return 0;
> +}
> +
> +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 < CPU_CGROUP_STAT_NSTATS; 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;
> +}
> +
>  #ifdef CONFIG_RT_GROUP_SCHED
>  static ssize_t cpu_rt_runtime_write(struct cgroup *cgrp, struct cftype *cft,
>      s64 val)
> @@ -8295,6 +8387,10 @@ static struct cftype cpu_files[] = {
>    .write_u64 = cpu_rt_period_write_uint,
>   },
>  #endif
> + {
> +  .name = "stat",
> +  .read_map = cpu_cgroup_stats_show,
> + },
>  };
>
>  static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
> @@ -8304,6 +8400,7 @@ static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct
cgroup *cont)
>
>  struct cgroup_subsys cpu_cgroup_subsys = {
>   .name  = "cpu",
> + .initialize = cpu_cgroup_initialize,
>   .create  = cpu_cgroup_create,
>   .destroy = cpu_cgroup_destroy,
>   .can_attach = cpu_cgroup_can_attach,
> --
```

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v4
Posted by Balaji Rao on Mon, 12 May 2008 05:41:24 GMT
View Forum Message <> Reply to Message

On Monday 12 May 2008 08:24:45 am Li Zefan wrote:
> Balaji Rao wrote:
> > /*
> > @@ -3892,8 +3923,17 @@ 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));
> > + }
> > + }
>
> You should put this '}' after '#endif'
>
Yes, my mistake. Will fix it.
> > +#endif
> > else if (atomic_read(&rq->nr_iowait) > 0)
> > cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
> > else
> > @@ -8179,10 +8219,26 @@ static inline struct task_group *cgroup_tg(struct
> > cgroup *cgrp) struct task_group, css);
> > }
> >
> > +static void cpu_cgroup_initialize(int early)
> > +{
> > + int i;
> > + struct cpu_cgroup_stat *stat;
> > +
> > + if (!early) {
> > + 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;
> > + }
> > +}
> > +

> > static struct cgroup_subsys_state *
> > cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
> > {
> >   struct task_group *tg;
> > + int i;
> >
> >   if (!cgrp->parent) {
> >   /* This is early initialization for the top cgroup */
> > @@ -8198,6 +8254,10 @@ 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);
> > +
>
> I guess you forgot to free those things in cpu_cgroup_destroy().
Oh. Sorry, I totally missed it.
>
> >   /* Bind the cgroup to task_group object we just created */

--
Warm Regards,

Balaji Rao
Dept. of Mechanical Engineering
NITK

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

---

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v4
Posted by akpm on Mon, 12 May 2008 23:48:33 GMT

View Forum Message <> Reply to Message

On Mon, 12 May 2008 01:48:37 +0530
Balaji Rao <balajirrao@gmail.com> wrote:

> Hi Andrew,
>
> Here's a version that uses percpu_counters and which actually works. The
> only evil it contains is the check,
>
> if (percpu_counter_ready) {
>   ..

> }

There's no instance of "percpu_counter_ready" in the patch so I'm a bit
stumped.

> @@ -3837,6 +3858,16 @@ 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
> }
>
> /*
> @@ -3892,8 +3923,17 @@ 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));
> + }
> + }
> +#endif

I'd suggest that the above be turned into calls to a helper function
which is a no-op if !CONFIG_CGROUP_SCHED.

Subject: Re: [RFC][-mm] Simple stats for cpu resource controller v4
Posted by Balaji Rao on Tue, 13 May 2008 03:30:27 GMT
View Forum Message <> Reply to Message

On Tuesday 13 May 2008 05:18:33 am Andrew Morton wrote:
> On Mon, 12 May 2008 01:48:37 +0530
>
> Balaji Rao <balajirrao@gmail.com> wrote:
> > Hi Andrew,
> >
> > Here's a version that uses percpu_counters and which actually works. The
> > only evil it contains is the check,
> >
> > if (percpu_counter_ready) {
> > ..
> > }
>
> There's no instance of "percpu_counter_ready" in the patch so I'm a bit
> stumped.
>
Here it is. 'stat' can tell us if percpu_counters are ready or not.

+static void __cpu_cgroup_stat_add(struct cpu_cgroup_stat *stat,

+{

+}
+#endif

It's here as well.

+static s64 cpu_cgroup_read_stat(struct cpu_cgroup_stat *stat,

+{

+

+}
+
> > @@ -3837,6 +3858,16 @@ 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
> > }
> >
> > /*
> > @@ -3892,8 +3923,17 @@ 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));
> > + }
> > + }
> > +#endif
>
> I'd suggest that the above be turned into calls to a helper function
> which is a no-op if !CONFIG_CGROUP_SCHED.

OK. Will do.

--
Warm Regards,

Balaji Rao
Dept. of Mechanical Engineering
NITK

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