Subject: [PATCH v2 0/5] per-cgroup /proc/stat statistics
Posted by Glauber Costa on Mon, 09 Apr 2012 22:25:10 GMT
View Forum Message <> Reply to Message

Hi,

This patch aims at allowing userspace to recreate the most important
contents of /proc/stat per-cgroup. It exports the data needed for it
from the guts of the scheduler, and then anyone can parse it and
present it to a container in a meaningful way. Again, the kernel won't
get involved in this directly.

Part of it will come from the cpu cgroup. Another part, from the cpuacct
cgroup. Data is exported in cgroup files stat_percpu. They are just like
the normal stat files, but with a cpuXXX value before the actual data
field. As so, they are also extensible. So if anyone wants to give a
shot at values I am currently ignoring (as iowait) in the future, we
at least won't have a format problem.

Let me know what you think.

Glauber Costa (5):
  measure exec_clock for rt sched entities
  account guest time per-cgroup as well.
  record nr_switches per task_group
  expose fine-grained per-cpu data for cpuacct stats
  expose per-taskgroup schedstats in cgroup

 kernel/sched/core.c | 214 ++--
 kernel/sched/fair.c | 27 ++++++-
 kernel/sched/rt.c | 5 +
 kernel/sched/sched.h | 6 ++
 4 files changed, 245 insertions(+), 7 deletions(-)

--
1.7.7.6

Subject: [PATCH v2 1/5] measure exec_clock for rt sched entities
Posted by Glauber Costa on Mon, 09 Apr 2012 22:25:11 GMT
View Forum Message <> Reply to Message

For simetry with the cfq tasks, measure exec_clock for the rt
sched entities (rt_se).

This can be used in a number of fashions. For instance, to
compute total cpu usage in a cgroup that is generated by
rt tasks.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 kernel/sched/rt.c | 5 +++++
 kernel/sched/sched.h | 1 +
 2 files changed, 6 insertions(+), 0 deletions(-)

diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
index 44af55e..02869a9 100644
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -919,6 +919,11 @@ static void update_curr_rt(struct rq *rq)

  sched_rt_avg_update(rq, delta_exec);

+ for_each_sched_rt_entity(rt_se) {
+ rt_rq = rt_rq_of_se(rt_se);
+ schedstat_add(rt_rq, exec_clock, delta_exec);
+ }
+
  if (!rt_bandwidth_enabled())
   return;

diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index fb3acba..b8bcd147 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -295,6 +295,7 @@ struct rt_rq {
  struct plist_head pushable_tasks;
 #endif
  int rt_throttled;
+ u64 exec_clock;
  u64 rt_time;
  u64 rt_runtime;
  /* Nests inside the rq lock: */
--
1.7.7.6

Subject: [PATCH v2 2/5] account guest time per-cgroup as well.
Posted by Glauber Costa on Mon, 09 Apr 2012 22:25:12 GMT

View Forum Message <> Reply to Message

We already track multiple tick statistics per-cgroup, using
the task_group_account_field facility. This patch accounts
guest_time in that manner as well.

Signed-off-by: Glauber Costa <glommer@parallels.com>

```
---
 kernel/sched/core.c |   10 ++++++------
 1 files changed, 4 insertions(+), 6 deletions(-)

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 4603b9d..1cfb7f0 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -2690,8 +2690,6 @@ void account_user_time(struct task_struct *p, cputime_t cputime,
 static void account_guest_time(struct task_struct *p, cputime_t cputime,
                cputime_t cputime_scaled)
 {
- u64 *cpustat = kcpustat_this_cpu->cpustat;
-
   /* Add guest time to process. */
   p->utime += cputime;
   p->utimescaled += cputime_scaled;
@@ -2700,11 +2698,11 @@ static void account_guest_time(struct task_struct *p, cputime_t
cputime,

   /* Add guest time to cpustat. */
   if (TASK_NICE(p) > 0) {
-  cpustat[CPUTIME_NICE] += (__force u64) cputime;
-  cpustat[CPUTIME_GUEST_NICE] += (__force u64) cputime;
+  task_group_account_field(p, CPUTIME_NICE, (__force u64) cputime);
+  task_group_account_field(p, CPUTIME_GUEST, (__force u64) cputime);
   } else {
-  cpustat[CPUTIME_USER] += (__force u64) cputime;
-  cpustat[CPUTIME_GUEST] += (__force u64) cputime;
+  task_group_account_field(p, CPUTIME_USER, (__force u64) cputime);
+  task_group_account_field(p, CPUTIME_GUEST, (__force u64) cputime);
   }
 }

--
1.7.7.6
```

Subject: [PATCH v2 3/5] record nr_switches per task_group
Posted by Glauber Costa on Mon, 09 Apr 2012 22:25:13 GMT
View Forum Message <> Reply to Message

In the interest of providing a per-cgroup figure of common statistics,
this patch adds a nr_switches counter to each group runqueue (both cfs
and rt).

To avoid impact on schedule(), we don't walk the tree at stat gather
time. This is because schedule() is called much more frequently than

the tick functions, in which we do walk the tree.

When this figure needs to be read (different patch), we will
aggregate them at read time.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 kernel/sched/core.c | 32 ++++++++++++++++++++++++++++++++
 kernel/sched/sched.h | 3 +++
 2 files changed, 35 insertions(+), 0 deletions(-)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 1cfb7f0..1ee3772 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -3168,6 +3168,37 @@ pick_next_task(struct rq *rq)
 }

 /*
+ * For all other data, we do a tree walk at the time of
+ * gathering. We want, however, to minimize the impact over schedule(),
+ * because... well... it's schedule().
+ *
+ * Therefore we only gather for the current cgroup, and walk the tree
+ * at read time
+ */
+static void update_switches_task_group(struct rq *rq,
+           struct task_struct *prev,
+           struct task_struct *next)
+{
+#ifdef CONFIG_CGROUP_SCHED
+ int cpu = cpu_of(rq);
+
+ if (rq->curr_tg == &root_task_group)
+  goto out;
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ if (prev->sched_class == &fair_sched_class)
+  rq->curr_tg->cfs_rq[cpu]->nr_switches++;
+#endif
+#ifdef CONFIG_RT_GROUP_SCHED
+ if (prev->sched_class == &rt_sched_class)
+  rq->curr_tg->rt_rq[cpu]->nr_switches++;
+#endif
+out:
+ rq->curr_tg = task_group(next);
+#endif
+}
```

```
+
+/*
  * __schedule() is the main scheduler function.
  */
 static void __sched __schedule(void)
@@ -3230,6 +3261,7 @@ need_resched:
   rq->curr = next;
   ++*switch_count;

+  update_switches_task_group(rq, prev, next);
   context_switch(rq, prev, next); /* unlocks the rq */
   /*
    * The context switch have flipped the stack from under us
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index b8bcd147..3b300f3 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -224,6 +224,7 @@ struct cfs_rq {

 #ifdef CONFIG_FAIR_GROUP_SCHED
  struct rq *rq; /* cpu runqueue to which this cfs_rq is attached */
+ u64 nr_switches;

  /*
   * leaf cfs_rqs are those that hold tasks (lowest schedulable entity in
@@ -307,6 +308,7 @@ struct rt_rq {
  struct rq *rq;
  struct list_head leaf_rt_rq_list;
  struct task_group *tg;
+ u64 nr_switches;
 #endif
 };

@@ -389,6 +391,7 @@ struct rq {
  unsigned long nr_uninterruptible;

  struct task_struct *curr, *idle, *stop;
+ struct task_group *curr_tg;
  unsigned long next_balance;
  struct mm_struct *prev_mm;

--
1.7.7.6
```

Subject: [PATCH v2 4/5] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Mon, 09 Apr 2012 22:25:14 GMT

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>
---
 kernel/sched/core.c | 34 ++++++++++++++++++++++++++++++++++
 1 files changed, 34 insertions(+), 0 deletions(-)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 1ee3772..52bae67 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -8186,6 +8186,35 @@ 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);
+ }
+
+ return 0;
+}
+
 static struct cftype files[] = {
  {
   .name = "usage",
@@ -8200,6 +8229,11 @@ static struct cftype files[] = {
   .name = "stat",
   .read_map = cpuacct_stats_show,
  },
+ {
+  .name = "stat_percpu",
+  .read_map = cpuacct_stats_percpu_show,
+ },
+
 };

 static int cpuacct_populate(struct cgroup_subsys *ss, struct cgroup *cgrp)
--
1.7.7.6
```

Subject: [PATCH v2 5/5] expose per-taskgroup schedstats in cgroup
Posted by Glauber Costa on Mon, 09 Apr 2012 22:25:15 GMT
View Forum Message <> Reply to Message

This patch aims at exposing stat information per-cgroup, such as:
 * idle time,
 * iowait time,
 * steal time,
 * # context switches
and friends. The ultimate goal is to be able to present a per-container view of
/proc/stat inside a container. With this patch, everything that is needed to do
that is in place, except for number of tasks.

For most of the data, I achieve that by hooking into the schedstats framework,
so although the overhead of that is prone to discussion, I am not adding anything,
but reusing what's already there instead. The exception being that the data is
now computed and stored in non-task se's as well, instead of entity_is_task() branches.
However, I expect this to be minimum comparing to the alternative of adding new
hierarchy walks. Those are kept intact.

The format of the new file added is the same as the one recently
introduced for cpuacct:

```
 cpu0.idle X
 cpu0.steal Y
 ...
 cpu1.idle X1
 cpu1.steal Y1
 ...
```

Signed-off-by: Glauber Costa <glommer@parallels.com>

```
 kernel/sched/core.c  |  138 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/sched/fair.c  |   27 +++++++++-
 kernel/sched/sched.h |    2 +
 3 files changed, 166 insertions(+), 1 deletions(-)
```

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 52bae67..e7d47c9 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -7964,6 +7964,131 @@ static u64 cpu_rt_period_read_uint(struct cgroup *cgrp, struct cftype
*cft)
 }
 #endif /* CONFIG_RT_GROUP_SCHED */

+#ifdef CONFIG_SCHEDSTATS
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+#define fair_rq(field, tg, i)  tg->cfs_rq[i]->field
+#else
+#define fair_rq(field, tg, i)  0
+#endif
+
+#ifdef CONFIG_RT_GROUP_SCHED
+#define rt_rq(field, tg, i)  tg->rt_rq[i]->field
+#else
+#define rt_rq(field, tg, i)  0
+#endif
+
```

```
+struct nr_switches_data {
+ u64 switches;
+ int cpu;
+};
+
+static int nr_switches_walker(struct task_group *tg, void *data)
+{
+ struct nr_switches_data *switches = data;
+ int cpu = switches->cpu;
+
+ switches->switches += fair_rq(nr_switches, tg, cpu) +
+         rt_rq(nr_switches, tg, cpu);
+ return 0;
+}
+
+static u64 tg_nr_switches(struct task_group *tg, int cpu)
+{
+ if (tg != &root_task_group) {
+  struct nr_switches_data data = {
+   .switches = 0,
+   .cpu = cpu,
+  };
+
+  rcu_read_lock();
+  walk_tg_tree_from(tg, nr_switches_walker, tg_nop, &data);
+  rcu_read_unlock();
+  return data.switches;
+ }
+
+ return cpu_rq(cpu)->nr_switches;
+}
+
+static u64 tg_nr_running(struct task_group *tg, int cpu)
+{
+ /*
+  * because of autogrouped groups in root_task_group, the
+  * following does not hold.
+  */
+ if (tg != &root_task_group)
+  return rt_rq(rt_nr_running, tg, cpu) + fair_rq(nr_running, tg, cpu);
+
+ return cpu_rq(cpu)->nr_running;
+}
+
+static u64 tg_idle(struct task_group *tg, int cpu)
+{
+ u64 val;
+
```

```
+ if (tg != &root_task_group) {
+  val = cfs_read_sleep(tg->se[cpu]);
+  /* If we have rt tasks running, we're not really idle */
+  val -= rt_rq(exec_clock, tg, cpu);
+ } else
+  /*
+   * There are many errors here that we are accumulating.
+   * However, we only provide this in the interest of having
+   * a consistent interface for all cgroups. Everybody
+   * probing the root cgroup should be getting its figures
+   * from system-wide files as /proc/stat. That would be faster
+   * to begin with...
+   *
+   * Ditto for steal.
+   */
+  val = kcpustat_cpu(cpu).cpustat[CPUTIME_IDLE] * TICK_NSEC;
+
+ return val;
+}
+
+static u64 tg_steal(struct task_group *tg, int cpu)
+{
+ u64 val;
+
+ if (tg != &root_task_group)
+  val = cfs_read_wait(tg->se[cpu]);
+ else
+  val = kcpustat_cpu(cpu).cpustat[CPUTIME_STEAL] * TICK_NSEC;
+
+ return val;
+}
+
+static int cpu_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
+     struct cgroup_map_cb *cb)
+{
+ struct task_group *tg = cgroup_tg(cgrp);
+ int cpu;
+ /*
+  * should be enough to hold:
+  * "cpu" (len = 3)
+  * "nr_switches" (len = 11, biggest string so far
+  * 4 bytes for the cpu number, up to 9999 cpus
+  * dot character and NULL termination,
+  *
+  * and still be small enough for the stack
+  */
+ char name[24];
+
```

```
+ for_each_online_cpu(cpu) {
+  snprintf(name, sizeof(name), "cpu%d.idle", cpu);
+  cb->fill(cb, name, tg_idle(tg, cpu));
+  snprintf(name, sizeof(name), "cpu%d.steal", cpu);
+  cb->fill(cb, name, tg_steal(tg, cpu));
+  snprintf(name, sizeof(name), "cpu%d.nr_switches", cpu);
+  cb->fill(cb, name, tg_nr_switches(tg, cpu));
+  snprintf(name, sizeof(name), "cpu%d.nr_running", cpu);
+  cb->fill(cb, name, tg_nr_running(tg, cpu));
+ }
+
+ return 0;
+}
+#endif
+
 static struct cftype cpu_files[] = {
 #ifdef CONFIG_FAIR_GROUP_SCHED
  {
@@ -7971,6 +8096,19 @@ static struct cftype cpu_files[] = {
   .read_u64 = cpu_shares_read_u64,
   .write_u64 = cpu_shares_write_u64,
  },
+/*
+ * In theory, those could be done using the rt tasks as a basis
+ * as well. Since we're interested in figures like idle, iowait, etc
+ * for the whole cgroup, the results should be the same.
+ * But that only complicates the code, and I doubt anyone using !FAIR_GROUP_SCHED
+ * is terribly interested in those.
+ */
+#ifdef CONFIG_SCHEDSTATS
+ {
+  .name = "stat_percpu",
+  .read_map = cpu_stats_percpu_show,
+ },
+#endif
 #endif
 #ifdef CONFIG_CFS_BANDWIDTH
  {
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 0d97ebd..895dcf4 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -719,6 +719,30 @@ update_stats_wait_start(struct cfs_rq *cfs_rq, struct sched_entity *se)
  schedstat_set(se->statistics.wait_start, rq_of(cfs_rq)->clock);
 }

+#ifdef CONFIG_SCHEDSTATS
+u64 cfs_read_sleep(struct sched_entity *se)
```

```
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.sum_sleep_runtime;
+
+ if (!se->statistics.sleep_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.sleep_start;
+}
+
+u64 cfs_read_wait(struct sched_entity *se)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.wait_sum;
+
+ if (!se->statistics.wait_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.wait_start;
+}
+#endif
+
 /*
  * Task is being enqueued - update stats:
  */
@@ -1182,7 +1206,8 @@ dequeue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se, int flags)
     se->statistics.sleep_start = rq_of(cfs_rq)->clock;
    if (tsk->state & TASK_UNINTERRUPTIBLE)
     se->statistics.block_start = rq_of(cfs_rq)->clock;
-  }
+  } else
+   se->statistics.sleep_start = rq_of(cfs_rq)->clock;
 #endif
  }

diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index 3b300f3..c90a0d2 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -1150,6 +1150,8 @@ extern void init_rt_rq(struct rt_rq *rt_rq, struct rq *rq);
 extern void unthrottle_offline_cfs_rqs(struct rq *rq);

 extern void account_cfs_bandwidth_used(int enabled, int was_enabled);
+extern u64 cfs_read_sleep(struct sched_entity *se);
+extern u64 cfs_read_wait(struct sched_entity *se);

 #ifdef CONFIG_NO_HZ
 enum rq_nohz_flag_bits {
```

--
1.7.7.6

Subject: Re: [PATCH v2 4/5] expose fine-grained per-cpu data for cpuacct stats
Posted by Sha Zhengju on Wed, 18 Apr 2012 12:30:39 GMT
View Forum Message <> Reply to Message

_

On Mon, Apr 9, 2012 at 6:25 PM, 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
> ...
>

Why not show the all-cpu data together with the per-cpu one? I think
the total one
is an usual concern in most cases.

> Signed-off-by: Glauber Costa <glommer@parallels.com>
>
> kernel/sched/core.c | 34 ++++++++++++++++++++++++++++++++++
> 1 files changed, 34 insertions(+), 0 deletions(-)
>
> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 1ee3772..52bae67 100644
> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -8186,6 +8186,35 @@ 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);
> +       }
> +
> +       return 0;
> +}
> +
>  static struct cftype files[] = {
>         {
>                 .name = "usage",
> @@ -8200,6 +8229,11 @@ static struct cftype files[] = {
>                 .name = "stat",
>                 .read_map = cpuacct_stats_show,
>         },
> +       {
> +               .name = "stat_percpu",
> +               .read_map = cpuacct_stats_percpu_show,
> +       },
> +
>  };
>
>  static int cpuacct_populate(struct cgroup_subsys *ss, struct cgroup *cgrp)
> --
```

> 1.7.7.6
>

Subject: Re: [PATCH v2 5/5] expose per-taskgroup schedstats in cgroup
Posted by Sha Zhengju on Wed, 18 Apr 2012 14:44:25 GMT
View Forum Message <> Reply to Message

On Mon, Apr 9, 2012 at 6:25 PM, Glauber Costa <glommer@parallels.com> wrote:
> This patch aims at exposing stat information per-cgroup, such as:
> * idle time,
> * iowait time,
> * steal time,
> * # context switches
> and friends. The ultimate goal is to be able to present a per-container view of
> /proc/stat inside a container. With this patch, everything that is needed to do
> that is in place, except for number of tasks.
>
> For most of the data, I achieve that by hooking into the schedstats framework,
> so although the overhead of that is prone to discussion, I am not adding anything,
> but reusing what's already there instead. The exception being that the data is
> now computed and stored in non-task se's as well, instead of entity_is_task() branches.
> However, I expect this to be minimum comparing to the alternative of adding new
> hierarchy walks. Those are kept intact.
>
> The format of the new file added is the same as the one recently
> introduced for cpuacct:
>
> cpu0.idle X
> cpu0.steal Y
> ...
> cpu1.idle X1
> cpu1.steal Y1
> ...
>

You define the idle time as the sum of task's sleeping time which i
think it needs to
discuss. It changes the current meaning of idle time which might be confusing
and can not reflect the the actual cpu busy-idle situation. IMHO, idle
time can just
be the true system value. Personally I prefer to your last version in
the way of computing
idle time (http://thread.gmane.org/gmane.linux.kernel/1194838). And
iowait can be
computed in the similar way.

As to steal time, "Steal time is the percentage of time a virtual CPU

waits for a real
CPU while the hypervisor is servicing another virtual processor".
Speaking from the
point of view of resource controlling(isolation), cgroup is a
lightweight method towards
virtualiztion, so I think obeying its primitive meaning is more
appropriate: the time not
servicing me including time stolen by the tasks of other cgroup.

Subject: Re: [PATCH v2 5/5] expose per-taskgroup schedstats in cgroup
Posted by Sha Zhengju on Wed, 18 Apr 2012 14:57:12 GMT
View Forum Message <> Reply to Message

On Wed, Apr 18, 2012 at 10:44 PM, Sha Zhengju <handai.szj@gmail.com> wrote:
> On Mon, Apr 9, 2012 at 6:25 PM, Glauber Costa <glommer@parallels.com> wrote:
>> This patch aims at exposing stat information per-cgroup, such as:
>> * idle time,
>> * iowait time,
>> * steal time,
>> * # context switches
>> and friends. The ultimate goal is to be able to present a per-container view of
>> /proc/stat inside a container. With this patch, everything that is needed to do
>> that is in place, except for number of tasks.
>>
>> For most of the data, I achieve that by hooking into the schedstats framework,
>> so although the overhead of that is prone to discussion, I am not adding anything,
>> but reusing what's already there instead. The exception being that the data is
>> now computed and stored in non-task se's as well, instead of entity_is_task() branches.
>> However, I expect this to be minimum comparing to the alternative of adding new
>> hierarchy walks. Those are kept intact.
>>
>> The format of the new file added is the same as the one recently
>> introduced for cpuacct:
>>
>> cpu0.idle X
>> cpu0.steal Y
>> ...
>> cpu1.idle X1
>> cpu1.steal Y1
>> ...
>>
>
> You define the idle time as the sum of task's sleeping time which i
> think it needs to
> discuss. It changes the current meaning of idle time which might be confusing
> and can not reflect the the actual cpu busy-idle situation. IMHO, idle
> time can just

> be the true system value. Personally I prefer to your last version in
> the way of computing
> idle time (http://thread.gmane.org/gmane.linux.kernel/1194838). And
> iowait can be
> computed in the similar way.
>
> As to steal time, "Steal time is the percentage of time a virtual CPU
> waits for a real
> CPU while the hypervisor is servicing another virtual processor".
> Speaking from the
> point of view of resource controlling(isolation), cgroup is a
> lightweight method towards
> virtualiztion, so I think obeying its primitive meaning is more
> appropriate: the time not
> servicing me including time stolen by the tasks of other cgroup.

I have a different version of exposing per-cgroup /proc/stat (parts of
them come from your patchset :)). I can sent them out if necessary.

Thanks,
Sha

Subject: Re: [PATCH v2 4/5] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 18 Apr 2012 16:14:48 GMT
View Forum Message <> Reply to Message

On 04/18/2012 09:30 AM, Sha Zhengju wrote:
> On Mon, Apr 9, 2012 at 6:25 PM, 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
>> > ...
>> >
> Why not show the all-cpu data together with the per-cpu one? I think
> the total one
> is an usual concern in most cases.
>
Because that is a trivial operation that can be done in userspace.

In general, I see no value in formatting this file any further if we'll
have to get to userspace for the final solution anyway.

Subject: Re: [PATCH v2 5/5] expose per-taskgroup schedstats in cgroup
Posted by Glauber Costa on Wed, 18 Apr 2012 16:24:37 GMT
View Forum Message <> Reply to Message

>
> You define the idle time as the sum of task's sleeping time which i
> think it needs to
> discuss.

Where is it done ?

Idle time here is measured as the time between enqueue_sleeper() and
the group being put back in the rq.
But note it is enqueue sleeper for the group, not any tasks.

cfs will call this callback until it finds anything that is running
(task or not a task).

Maybe I made some mistake in the code - and in this case, please point
out - but that's the idea.

> IMHO, idle
> time can just
> be the true system value. Personally I prefer to your last version in
> the way of computing
> idle time (http://thread.gmane.org/gmane.linux.kernel/1194838). And
> iowait can be
> computed in the similar way.

No. The idea that idle time can only be true system-wide is wrong. As a
matter of fact, that first series of mine is totally wrong wrt that (and
then I changed).

A cgroup is idle when none of its tasks are in the runqueue. What is the

problem that you see with this?

As for iowait, that one seemed a bit trickier, so we decided to leave it
out at least for now.

>
> As to steal time, "Steal time is the percentage of time a virtual CPU
> waits for a real
> CPU while the hypervisor is servicing another virtual processor".
> Speaking from the
> point of view of resource controlling(isolation), cgroup is a
> lightweight method towards
> virtualiztion, so I think obeying its primitive meaning is more
> appropriate: the time not
> servicing me including time stolen by the tasks of other cgroup.

And that's exactly what I've done.

Steal time is runqueue time, until you are chosen to run.

In a summary: If you are not running, you can be either idle or stolen.
if you are in the runqueue, you are stolen.
If you are not, you are idle.

Subject: Re: [PATCH v2 5/5] expose per-taskgroup schedstats in cgroup
Posted by Sha Zhengju on Thu, 19 Apr 2012 13:30:04 GMT
View Forum Message <> Reply to Message

On 04/19/2012 12:24 AM, Glauber Costa wrote:
>
>>
>> You define the idle time as the sum of task's sleeping time which i
>> think it needs to
>> discuss.
>
> Where is it done ?
>
> Idle time here is measured as the time between enqueue_sleeper() and
> the group being put back in the rq.
> But note it is enqueue sleeper for the group, not any tasks.
>

Sorry, I still do not catch the point. In enqueue_sleeper(), it sums up
the sleep
time to se.statistics since dequeue_sleeper(), and then put back to rq.
Here do
you mean idle time is measured as the time between dequeue_sleeper() and

enqueue_sleeper()? But it's still the sum of sleeping time of the
group's task?
Not cfs expert. If I've miss something, please feel free to point it
out. :-)

> cfs will call this callback until it finds anything that is running
> (task or not a task).
>
> Maybe I made some mistake in the code - and in this case, please point
> out - but that's the idea.
>
>> IMHO, idle
>> time can just
>> be the true system value. Personally I prefer to your last version in
>> the way of computing
>> idle time (http://thread.gmane.org/gmane.linux.kernel/1194838). And
>> iowait can be
>> computed in the similar way.
>
> No. The idea that idle time can only be true system-wide is wrong. As a
> matter of fact, that first series of mine is totally wrong wrt that
> (and then I changed).
>
> A cgroup is idle when none of its tasks are in the runqueue. What is
> the problem that you see with this?

Actually, both idle and steal are the time when the group don't work.
IMO, i'd like to contribute
the real cpu idle time to a group's idle, and let the time cpu servicing
for other group to be steal time.
For example, suppose that 2 tasks(groups) are sharing one cpu and #1
keep running while #2 keep sleeping,
in your way: #1(idle)=0, #1(steal)=0; #2(idle)=100%, #2(steal)=0;
in my way: #1(idle)=0, #1(steal)=0; #2(idle)=0,   #2(steal)=100%;
IMHO, our opinions diverge from the meaning of "idle". But both idle and
steal can be get from cpuacct
in my way without involving in cpu controller.

>
> As for iowait, that one seemed a bit trickier, so we decided to leave
> it out at least for now.
>
>>
>> As to steal time, "Steal time is the percentage of time a virtual CPU
>> waits for a real
>> CPU while the hypervisor is servicing another virtual processor".
>> Speaking from the
>> point of view of resource controlling(isolation), cgroup is a

>> lightweight method towards
>> virtualiztion, so I think obeying its primitive meaning is more
>> appropriate: the time not
>> servicing me including time stolen by the tasks of other cgroup.
>
> And that's exactly what I've done.
>
> Steal time is runqueue time, until you are chosen to run.
>
> In a summary: If you are not running, you can be either idle or stolen.
> if you are in the runqueue, you are stolen.
> If you are not, you are idle.

Subject: Re: [PATCH v2 5/5] expose per-taskgroup schedstats in cgroup
Posted by Glauber Costa on Thu, 19 Apr 2012 15:00:37 GMT
View Forum Message <> Reply to Message

On 04/19/2012 10:30 AM, Sha Zhengju wrote:
> On 04/19/2012 12:24 AM, Glauber Costa wrote:
>>
>>>
>>> You define the idle time as the sum of task's sleeping time which i
>>> think it needs to
>>> discuss.
>>
>> Where is it done ?
>>
>> Idle time here is measured as the time between enqueue_sleeper() and
>> the group being put back in the rq.
>> But note it is enqueue sleeper for the group, not any tasks.
>>
>
> Sorry, I still do not catch the point. In enqueue_sleeper(), it sums up
> the sleep
> time to se.statistics since dequeue_sleeper(), and then put back to rq.
Yes.

> Here do
> you mean idle time is measured as the time between dequeue_sleeper() and
> enqueue_sleeper()?

In general, yes. In practice, when we read the field, a
dequeue_sleeper() may not yet have happened. So we need to sum whatever
is in the rq at the moment to it.

But it's still the sum of sleeping time of the
> group's task?

No.

cfs will walk the hierarchy up calling enqueue_sleeper until it finds a
group that is not sleeping. This means that enqueue_sleeper() will be
called when no tasks in the group are running. (actually, no subgroups,
since it is hierarchical).

Take a look at sched/core/fair.c

>>> IMHO, idle
>>> time can just
>>> be the true system value. Personally I prefer to your last version in
>>> the way of computing
>>> idle time (http://thread.gmane.org/gmane.linux.kernel/1194838). And
>>> iowait can be
>>> computed in the similar way.
>>
>> No. The idea that idle time can only be true system-wide is wrong. As a
>> matter of fact, that first series of mine is totally wrong wrt that
>> (and then I changed).
>>
>> A cgroup is idle when none of its tasks are in the runqueue. What is
>> the problem that you see with this?
>
> Actually, both idle and steal are the time when the group don't work.
The difference is why it doesn't work. If he doesn't want to work,
that's idle. If it can't work, that's steal time.

> IMO, i'd like to contribute
> the real cpu idle time to a group's idle, and let the time cpu servicing
> for other group to be steal time.

"contribute the real idle time to a group's idle" doesn't make any
sense. If all groups are idle, they all passed through
enqueue_sleeper(), and that time is already counted as idle. For all of
them.

About steal time: That's *exactly* what I am doing! When a group enters
the runqueue, it should run. If it doesn't run, that's because someone
else is running. Therefore, runqueue time == steal time.

> For example, suppose that 2 tasks(groups) are sharing one cpu and #1
> keep running while #2 keep sleeping,
> in your way: #1(idle)=0, #1(steal)=0; #2(idle)=100%, #2(steal)=0;
> in my way: #1(idle)=0, #1(steal)=0; #2(idle)=0, #2(steal)=100%;

Have you actually tested this?
It depends on what you mean by "keep sleeping". If you mean sleeping as

not having any work to do, of course it is idle time.

Believing this is steal time just because another group exists in the
system is just wrong.

> IMHO, our opinions diverge from the meaning of "idle".

Yes, I believe idle time is the time during which you are idle.

> But both idle and
> steal can be get from cpuacct
> in my way without involving in cpu controller.

How so? If you wait for the idle tick to happen, that will mean *ALL*
your groups are idle. And that is *not* how you measure idle time.

Idle time of a group of tasks, is the time during which none of the
tasks are running.

Subject: Re: [PATCH v2 0/5] per-cgroup /proc/stat statistics
Posted by Glauber Costa on Thu, 24 May 2012 09:10:26 GMT
View Forum Message <> Reply to Message

On 04/10/2012 02:25 AM, Glauber Costa wrote:
> Hi,
>
> This patch aims at allowing userspace to recreate the most important
> contents of /proc/stat per-cgroup. It exports the data needed for it
> from the guts of the scheduler, and then anyone can parse it and
> present it to a container in a meaningful way. Again, the kernel won't
> get involved in this directly.
>
> Part of it will come from the cpu cgroup. Another part, from the cpuacct
> cgroup. Data is exported in cgroup files stat_percpu. They are just like
> the normal stat files, but with a cpuXXX value before the actual data
> field. As so, they are also extensible. So if anyone wants to give a
> shot at values I am currently ignoring (as iowait) in the future, we
> at least won't have a format problem.
>
> Let me know what you think.
>
> Glauber Costa (5):
> measure exec_clock for rt sched entities
> account guest time per-cgroup as well.
> record nr_switches per task_group
> expose fine-grained per-cpu data for cpuacct stats
> expose per-taskgroup schedstats in cgroup

>
> kernel/sched/core.c | 214 ++--
> kernel/sched/fair.c | 27 ++++++-
> kernel/sched/rt.c | 5 +
> kernel/sched/sched.h | 6 ++
> 4 files changed, 245 insertions(+), 7 deletions(-)
>
Paul and the other scheduler folks:

Do you have a saying on this?

Thanks

Subject: Re: [PATCH v2 2/5] account guest time per-cgroup as well.
Posted by Paul Turner on Sat, 26 May 2012 04:44:58 GMT
View Forum Message <> Reply to Message

On 04/09/2012 03:25 PM, Glauber Costa wrote:
> In the interest of providing a per-cgroup figure of common statistics,
> this patch adds a nr_switches counter to each group runqueue (both cfs
> and rt).
>
> To avoid impact on schedule(), we don't walk the tree at stat gather
> time. This is because schedule() is called much more frequently than
> the tick functions, in which we do walk the tree.
>
> When this figure needs to be read (different patch), we will
> aggregate them at read time.
>
> Signed-off-by: Glauber Costa <glommer-bzQdu9zFT3WakBO8gow8eQ@public.gmane.org>
> ---
> kernel/sched/core.c | 32 ++++++++++++++++++++++++++++++++
> kernel/sched/sched.h | 3 +++
> 2 files changed, 35 insertions(+), 0 deletions(-)
>
> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 1cfb7f0..1ee3772 100644
> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -3168,6 +3168,37 @@ pick_next_task(struct rq *rq)
> }
>
> /*
> + * For all other data, we do a tree walk at the time of
> + * gathering. We want, however, to minimize the impact over schedule(),
> + * because... well... it's schedule().
> + *

> + * Therefore we only gather for the current cgroup, and walk the tree
> + * at read time
> + */
> +static void update_switches_task_group(struct rq *rq,
> +   struct task_struct *prev,
> +   struct task_struct *next)
> +{
> +#ifdef CONFIG_CGROUP_SCHED
> + int cpu = cpu_of(rq);
> +
> + if (rq->curr_tg == &root_task_group)
> + goto out;
> +
> +#ifdef CONFIG_FAIR_GROUP_SCHED
> + if (prev->sched_class == &fair_sched_class)
> + rq->curr_tg->cfs_rq[cpu]->nr_switches++;
> +#endif
> +#ifdef CONFIG_RT_GROUP_SCHED
> + if (prev->sched_class == &rt_sched_class)
> + rq->curr_tg->rt_rq[cpu]->nr_switches++;
> +#endif

With this approach why differentiate cfs vs rt? These could both just
be on the task_group.

This could then just be
if (prev != root_task_group)
  task_group(prev)->nr_switches++;

Which you could wrap in a nice static inline that disappears when
CONFIG_CGROUP_PROC_STAT isn't there

Another way to go about this would be to promote (demote?) nr_switches
to the sched_entity. At which point you know you only need to update
yours, and conditionally update your parents.

But.. that's still gross.. Hmm..

> +out:
> + rq->curr_tg = task_group(next);

If you're going to task_group every time anyway you might as well just
take it against prev -- then you don't have to cache rq->curr_tg?

Another way to do this would be:

On cfs_rq, rt_rq add:
  int prev_rq_nr_switches, nr_switches

On put_prev_prev_task_fair (against a task)
  cfs_rq_of(prev->se)->prev_rq_nr_switches = rq->nr_switches

On pick_next_task_fair:
  if (cfs_rq_of(prev->se)->prev_rq_nr_switches != rq->nr_switches)
    cfs_rq->nr_switches++;

On aggregating the value for read: +1 if prev_rq_nr_running != rq->nr_running
[And equivalent for sched_rt]

While this is no nicer (and fractionally more expensive but this is
never something we'd enable by default), it at least gets the goop out
of schedule().

> +#endif
> +}
> +
> +/*
>   * __schedule() is the main scheduler function.
>   */
> static void __sched __schedule(void)
> @@ -3230,6 +3261,7 @@ need_resched:
>   rq->curr = next;
>   ++*switch_count;
>
> + update_switches_task_group(rq, prev, next);
>   context_switch(rq, prev, next); /* unlocks the rq */
>   /*
>   * The context switch have flipped the stack from under us
> diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
> index b8bcd147..3b300f3 100644
> --- a/kernel/sched/sched.h
> +++ b/kernel/sched/sched.h
> @@ -224,6 +224,7 @@ struct cfs_rq {
>
> #ifdef CONFIG_FAIR_GROUP_SCHED
>   struct rq *rq; /* cpu runqueue to which this cfs_rq is attached */
> + u64 nr_switches;
>
>   /*
>   * leaf cfs_rqs are those that hold tasks (lowest schedulable entity in
> @@ -307,6 +308,7 @@ struct rt_rq {
>   struct rq *rq;
>   struct list_head leaf_rt_rq_list;
>   struct task_group *tg;
> + u64 nr_switches;
> #endif

> };
>
> @@ -389,6 +391,7 @@ struct rq {
> unsigned long nr_uninterruptible;
>
> struct task_struct *curr, *idle, *stop;
> + struct task_group *curr_tg;

This is a little gross. Is task_group even defined without CONFIG_CGROUP_SCHED?

> unsigned long next_balance;
> struct mm_struct *prev_mm;
>

Subject: Re: [PATCH v2 2/5] account guest time per-cgroup as well.
Posted by Glauber Costa on Mon, 28 May 2012 09:03:19 GMT
View Forum Message <> Reply to Message

On 05/26/2012 08:44 AM, Paul Turner wrote:
> On 04/09/2012 03:25 PM, Glauber Costa wrote:
>> In the interest of providing a per-cgroup figure of common statistics,
>> this patch adds a nr_switches counter to each group runqueue (both cfs
>> and rt).
>>
>> To avoid impact on schedule(), we don't walk the tree at stat gather
>> time. This is because schedule() is called much more frequently than
>> the tick functions, in which we do walk the tree.
>>
>> When this figure needs to be read (different patch), we will
>> aggregate them at read time.
>>
>> Signed-off-by: Glauber Costa<glommer-bzQdu9zFT3WakBO8gow8eQ@public.gmane.org>
>> ---
>> kernel/sched/core.c | 32 ++++++++++++++++++++++++++++++++
>> kernel/sched/sched.h | 3 +++
>> 2 files changed, 35 insertions(+), 0 deletions(-)
>>
>> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
>> index 1cfb7f0..1ee3772 100644
>> --- a/kernel/sched/core.c
>> +++ b/kernel/sched/core.c
>> @@ -3168,6 +3168,37 @@ pick_next_task(struct rq *rq)
>> }
>>
>> /*
>> + * For all other data, we do a tree walk at the time of
>> + * gathering. We want, however, to minimize the impact over schedule(),

```
>> + * because... well... it's schedule().
>> + *
>> + * Therefore we only gather for the current cgroup, and walk the tree
>> + * at read time
>> + */
>> +static void update_switches_task_group(struct rq *rq,
>> +           struct task_struct *prev,
>> +           struct task_struct *next)
>> +{
>> +#ifdef CONFIG_CGROUP_SCHED
>> + int cpu = cpu_of(rq);
>> +
>> + if (rq->curr_tg ==&root_task_group)
>> +  goto out;
>> +
>> +#ifdef CONFIG_FAIR_GROUP_SCHED
>> + if (prev->sched_class ==&fair_sched_class)
>> +  rq->curr_tg->cfs_rq[cpu]->nr_switches++;
>> +#endif
>> +#ifdef CONFIG_RT_GROUP_SCHED
>> + if (prev->sched_class ==&rt_sched_class)
>> +  rq->curr_tg->rt_rq[cpu]->nr_switches++;
>> +#endif
>
> With this approach why differentiate cfs vs rt?  These could both just
> be on the task_group.
>
> This could then just be
> if (prev != root_task_group)
>     task_group(prev)->nr_switches++;
```

well, no. Then it needs to be an atomic update, or something like it. The runqueue is a percpu data, the task_group is not. That's why I choosed to use a rq (and the runqueues are separated between classes).

It all boils down to the fact that I wanted to avoid an atomic update in this path.

But if you think that would be okay, I could change it. Alternatively, I could come up with another percpu storage as well, since we're ultimately just reading it later (and for that we need to iterate on all cpus anyway).

> Which you could wrap in a nice static inline that disappears when
> CONFIG_CGROUP_PROC_STAT isn't there

That I can do.

>
> Another way to go about this would be to promote (demote?) nr_switches
> to the sched_entity. At which point you know you only need to update
> yours, and conditionally update your parents.

You mean the global one ?

Not sure it will work, because that always refer to the root cgroup...

> But.. that's still gross.. Hmm..
>
>> +out:
>> + rq->curr_tg = task_group(next);
>
> If you're going to task_group every time anyway you might as well just
> take it against prev -- then you don't have to cache rq->curr_tg?
>
> Another way to do this would be:
>
> On cfs_rq, rt_rq add:
>   int prev_rq_nr_switches, nr_switches
>
> On put_prev_prev_task_fair (against a task)
>   cfs_rq_of(prev->se)->prev_rq_nr_switches = rq->nr_switches
>
> On pick_next_task_fair:
>   if (cfs_rq_of(prev->se)->prev_rq_nr_switches != rq->nr_switches)
>   cfs_rq->nr_switches++;
>
> On aggregating the value for read: +1 if prev_rq_nr_running != rq->nr_running
> [And equivalent for sched_rt]
>
> While this is no nicer (and fractionally more expensive but this is
> never something we'd enable by default), it at least gets the goop out
> of schedule().

At first look this sounds a bit weird to me, but OTOH, this is how a lot
of the stuff is done... All the other statistics in the patch set are
collected this exact same way - because it draws from schedstats, that
always touch the specific rqs, so maybe this gain points for consistency.

I'll give it a shot.

BTW, this means that your first comment about merging cfq and rt is
basically to be disconsidered should I take this route, right ?

Subject: Re: [PATCH v2 2/5] account guest time per-cgroup as well.
Posted by Glauber Costa on Mon, 28 May 2012 13:26:25 GMT
View Forum Message <> Reply to Message

On 05/26/2012 08:44 AM, Paul Turner wrote:
> On 04/09/2012 03:25 PM, Glauber Costa wrote:
>> In the interest of providing a per-cgroup figure of common statistics,
>> this patch adds a nr_switches counter to each group runqueue (both cfs
>> and rt).
>>
>> To avoid impact on schedule(), we don't walk the tree at stat gather
>> time. This is because schedule() is called much more frequently than
>> the tick functions, in which we do walk the tree.
>>
>> When this figure needs to be read (different patch), we will
>> aggregate them at read time.
>>
>>

Paul,

How about the following patch instead?

It is still using the cfs_rq and rt_rq's structures, (this code actually only touches fair.c as a PoC, rt would be similar).

Tasks in the root cgroup (without an se->parent), will do a branch and exit. For the others, we accumulate here, and simplify the reader.

My reasoning for this, is based on the fact that all the se->parent relations should be cached by our recent call to put_prev_task (well, unless of course we have a really big chain)

This would incur a slightly higher context switch time for tasks inside a cgroup.

The reader (in a different patch) would then be the same as the others:

```
+static u64 tg_nr_switches(struct task_group *tg, int cpu)
+{
+ if (tg != &root_task_group)
+  return rt_rq(rt_nr_switches, tg, cpu)
                    +fair_rq(nr_switches, tg, cpu);
+
+ return cpu_rq(cpu)->nr_switches;
+}
```

I plan to measure this today, but an extra branch cost for the common case of a task in the root cgroup + O(depth) for tasks inside cgroups

may be acceptable, given the simplification it brings.

Let me know what you think.

File Attachments

1) alternative.patch, downloaded 459 times

Subject: Re: [PATCH v2 2/5] account guest time per-cgroup as well.
Posted by Glauber Costa on Tue, 29 May 2012 10:34:55 GMT
View Forum Message <> Reply to Message

On 05/28/2012 05:26 PM, Glauber Costa wrote:
>
> I plan to measure this today, but an extra branch cost for the common
> case of a task in the root cgroup + O(depth) for tasks inside cgroups
> may be acceptable, given the simplification it brings.
>
> Let me know what you think.

Numbers:

benchmark is hackbench -pipe 1 thread 4000

task sitting in the root cgroup
==========
Without this patch:
4.857700 (0.69 %)
With this patch:
4.828733 (0.55 %)

Difference between them: 0.59 %, very close to the standard deviation,
no real difference.

task sitting in a 3-level cgroup
==========
Without this patch
5.120867 (1.60 %)
With this patch
5.126267 (1.30 %)

Difference between them: 0.10 %, way within the standard deviation

Task sitting in a level-30 cgroup: (total crazy)
==========
Without this patch:
8.829385 (2.63 %)

With this patch:
9.347846 (2.25 %)

Difference is about 5.8 %, way out of the standard deviation, so it is really worse. But who uses 30-level hierarchy?

I believe depth-3 is close to a practical worst case, for the very majority of the workloads out there. Therefore I don't see the loop here as a big problem. It does degrade, but not in any use case that matters.
