Subject: [PATCH v3 0/6] per cgroup /proc/stat statistics
Posted by Glauber Costa on Wed, 30 May 2012 09:48:31 GMT
View Forum Message <> Reply to Message

Hi

This is a rework of my v2 series taking into account Paul
Turner's comments. All the series is the same except for
the patch that collets nr_switches that is now completely reworked.

It is out of schedule(), relegated to a scheduler hook. Only difference
from Paul's suggestion is that I added a new one, instead of messing with
put_prev_task + pick_next_task pairs. In the patchset that adds it,
I'll try to argue for my solution.

Hope this is acceptable, and please let me know of any other concerns.

v2:
* completely reworked nr_switches gathering
* separated per-se sleep_start to be more clear about it

Glauber Costa (6):
  measure exec_clock for rt sched entities
  account guest time per-cgroup as well.
  expose fine-grained per-cpu data for cpuacct stats
  add a new scheduler hook for context switch
  Also record sleep start for a task group
  expose per-taskgroup schedstats in cgroup

```
 include/linux/sched.h |    1 +
 kernel/sched/core.c   |  166 ++++++++++++++++++++++++++++++++++++++++++++++++--
 kernel/sched/fair.c   |   42 ++++++++++++-
 kernel/sched/rt.c     |   20 ++++++
 kernel/sched/sched.h  |    6 ++
 5 files changed, 228 insertions(+), 7 deletions(-)
```

--
1.7.10.2

Subject: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Glauber Costa on Wed, 30 May 2012 09:48:32 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>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/rt.c | 5 +++++
 kernel/sched/sched.h | 1 +
 2 files changed, 6 insertions(+)

```
diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
index c5565c3..30ee4e2 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 ba9dccf..cd2f1e1 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: */
--
1.7.10.2
```

Subject: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 09:48:34 GMT
View Forum Message <> Reply to Message

The cpuacct cgroup already exposes user and system numbers in a per-cgroup

fashion. But they are a summation along the whole group, not a per-cpu figure.
Also, they are coarse-grained version of the stats usually shown at places
like /proc/stat.

I want to have enough cgroup data to emulate the /proc/stat interface. To
achieve that, I am creating a new file "stat_percpu" that displays the
fine-grained per-cpu data. The original data is left alone.

The format of this file resembles the one found in the usual cgroup's stat
files. But of course, the fields will be repeated, one per cpu, and prefixed
with the cpu number.

Therefore, we'll have something like:

```
 cpu0.user X
 cpu0.system Y
 ...
 cpu1.user X1
 cpu1.system Y1
 ...
```

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/core.c | 33 +++++++++++++++++++++++++++++++++
 1 file changed, 33 insertions(+)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 220d416..4c1d7e9 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -8178,6 +8178,35 @@ static int cpuacct_stats_show(struct cgroup *cgrp, struct cftype *cft,
  return 0;
 }

+static inline void do_fill_cb(struct cgroup_map_cb *cb, struct cpuacct *ca,
+        char *str, int cpu, int index)
+{
+ char name[24];
+ struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
+
+ snprintf(name, sizeof(name), "cpu%d.%s", cpu, str);
+ cb->fill(cb, name, cputime64_to_clock_t(kcpustat->cpustat[index]));
+}
+
+static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
+        struct cgroup_map_cb *cb)
```

```
+{
+ struct cpuacct *ca = cgroup_ca(cgrp);
+ int cpu;
+
+ for_each_online_cpu(cpu) {
+  do_fill_cb(cb, ca, "user", cpu, CPUTIME_USER);
+  do_fill_cb(cb, ca, "nice", cpu, CPUTIME_NICE);
+  do_fill_cb(cb, ca, "system", cpu, CPUTIME_SYSTEM);
+  do_fill_cb(cb, ca, "irq", cpu, CPUTIME_IRQ);
+  do_fill_cb(cb, ca, "softirq", cpu, CPUTIME_SOFTIRQ);
+  do_fill_cb(cb, ca, "guest", cpu, CPUTIME_GUEST);
+  do_fill_cb(cb, ca, "guest_nice", cpu, CPUTIME_GUEST_NICE);
+ }
+
+ return 0;
+}
+
 static struct cftype files[] = {
  {
   .name = "usage",
@@ -8192,6 +8221,10 @@ static struct cftype files[] = {
   .name = "stat",
   .read_map = cpuacct_stats_show,
  },
+ {
+  .name = "stat_percpu",
+  .read_map = cpuacct_stats_percpu_show,
+ },
  { } /* terminate */
 };

--
1.7.10.2
```

Subject: [PATCH v3 4/6] add a new scheduler hook for context switch
Posted by Glauber Costa on Wed, 30 May 2012 09:48:35 GMT
View Forum Message <> Reply to Message

To be able to count the number of switches per-cgroup, and merging
Paul's hint that it would be better to do it in fair.c and rt.c,
I am introducing a new write-side walk through a new scheduler hook,
called at every context switch away from a task (prev).

Read-side is greatly simplified, and as I'll show, the performance impact
does not seem huge. First, aside from the function call, this walk is
O(depth), which is not likely to be huge (if it is, the performance
impact is indeed bad - but one can argue this is a good punishment)

Also, this walk is likely to be cache-hot, since it is at most the very
same loop done by put_prev_task, except it loops depth - 1 instead of depth
times. This is specially important not to hurt tasks in the root cgroup,
that will pay just a branch.

I am introducing a new hook, because the existing ones didn't seem
appropriate. The main possibilities would be put_prev_task and
pick_next_task.

With pick_next_task, there are two main problems:

1) first, the loop is only cache hot in pick_next_task if tasks actually
belong to the same class and group as prev. Depending on the workload,
this is possibly unlikely.

2) This is vulnerable to wrongdoings in accountings when exiting to idle.
Consider two groups A and B with a following pattern: A exits to idle,
but after that, B is scheduled. B, on the other hand, always get back
to itself when it yields to idle. This means that the to-idle transition
in A is never noted.

So because of that, I do believe that prev is the right point of that.

However, put_prev_task is called many times from multiple places,
and the logic to differentiate a context switch from another kind of
put would make a mess out of it.

On a 4-way x86_64, hackbench -pipe 1 process 4000 shows the following results:
- units are seconds to complete the whole benchmark
- percentual stdev for easier assesment

Task sitting in the root cgroup:
Without patchset: 4.857700 (0.69 %)
With patchset: 4.863700 (0.63 %)
Difference : 0.12 %

Task sitting in a 3-level depth cgroup:
Without patchset: 5.120867 (1.60 %)
With patchset: 5.113800 (0.41 %)
Difference : 0.13 %

Task sitting in a 30-level depth cgroup (totally crazy scenario):
Without patchset: 8.829385 (2.63 %)
With patchset: 9.975467 (2.80 %)
Difference : 12 %

For any sane use case, the user is unlikely to be nesting for much more than

3-levels. For that, and for the important case for most people, the difference
is inside the standard deviation and can be said to be negligible.

Although the patch does add a penalty, it only does that for
deeply nested scenarios (but those are already paying a 100 % penalty
against no-nesting even without the patchset!)

I hope this approach is acceptable.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 include/linux/sched.h | 1 +
 kernel/sched/core.c | 9 +++++++++
 kernel/sched/fair.c | 15 +++++++++++++++
 kernel/sched/rt.c | 15 +++++++++++++++
 kernel/sched/sched.h | 3 +++
 5 files changed, 43 insertions(+)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index f45c0b2..d28d6ec 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1084,6 +1084,7 @@ struct sched_class {

  struct task_struct * (*pick_next_task) (struct rq *rq);
  void (*put_prev_task) (struct rq *rq, struct task_struct *p);
+ void (*context_switch) (struct rq *rq, struct task_struct *p);

 #ifdef CONFIG_SMP
  int (*select_task_rq)(struct task_struct *p, int sd_flag, int flags);
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 4c1d7e9..db4f2c3 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -1894,6 +1894,14 @@ fire_sched_out_preempt_notifiers(struct task_struct *curr,

 #endif /* CONFIG_PREEMPT_NOTIFIERS */

+static void sched_class_context_switch(struct rq *rq, struct task_struct *prev)
+{
+#if defined(CONFIG_FAIR_GROUP_SCHED) || defined(CONFIG_RT_GROUP_SCHED)
+ if (prev->sched_class->context_switch)
+ prev->sched_class->context_switch(rq, prev);
+#endif
+}
+

```
 /**
  * prepare_task_switch - prepare to switch tasks
  * @rq: the runqueue preparing to switch
@@ -1911,6 +1919,7 @@ static inline void
 prepare_task_switch(struct rq *rq, struct task_struct *prev,
       struct task_struct *next)
 {
+ sched_class_context_switch(rq, prev);
  sched_info_switch(prev, next);
  perf_event_task_sched_out(prev, next);
  fire_sched_out_preempt_notifiers(prev, next);
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 940e6d1..c26fe38 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -2993,6 +2993,20 @@ static struct task_struct *pick_next_task_fair(struct rq *rq)
  return p;
 }

+static void context_switch_fair(struct rq *rq, struct task_struct *p)
+{
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ struct cfs_rq *cfs_rq;
+ struct sched_entity *se = &p->se;
+
+ while (se->parent) {
+  se = se->parent;
+  cfs_rq = group_cfs_rq(se);
+  cfs_rq->nr_switches++;
+ }
+#endif
+}
+
 /*
  * Account for a descheduled task:
  */
@@ -5255,6 +5269,7 @@ const struct sched_class fair_sched_class = {
  .check_preempt_curr = check_preempt_wakeup,

  .pick_next_task  = pick_next_task_fair,
+ .context_switch  = context_switch_fair,
  .put_prev_task  = put_prev_task_fair,

 #ifdef CONFIG_SMP
diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
index 30ee4e2..6f416e4 100644
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
```

```
@@ -1392,6 +1392,20 @@ static struct task_struct *pick_next_task_rt(struct rq *rq)
  return p;
 }

+static void context_switch_rt(struct rq *rq, struct task_struct *p)
+{
+#ifdef CONFIG_RT_GROUP_SCHED
+ struct sched_rt_entity *rt_se = &p->rt;
+ struct rt_rq *rt_rq;
+
+ while (rt_se->parent) {
+  rt_se = rt_se->parent;
+  rt_rq = group_rt_rq(rt_se);
+  rt_rq->rt_nr_switches++;
+ }
+#endif
+}
+
 static void put_prev_task_rt(struct rq *rq, struct task_struct *p)
 {
  update_curr_rt(rq);
@@ -2040,6 +2054,7 @@ const struct sched_class rt_sched_class = {
  .check_preempt_curr = check_preempt_curr_rt,

  .pick_next_task  = pick_next_task_rt,
+ .context_switch  = context_switch_rt,
  .put_prev_task  = put_prev_task_rt,

 #ifdef CONFIG_SMP
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index cd2f1e1..76f6839 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -237,6 +237,7 @@ struct cfs_rq {
  struct list_head leaf_cfs_rq_list;
  struct task_group *tg; /* group that "owns" this runqueue */

+ u64 nr_switches;
 #ifdef CONFIG_SMP
  /*
   *   h_load = weight * f(tg)
@@ -307,6 +308,8 @@ struct rt_rq {
  struct rq *rq;
  struct list_head leaf_rt_rq_list;
  struct task_group *tg;
+
+ u64 rt_nr_switches;
 #endif
```

};

--
1.7.10.2

Subject: [PATCH v3 5/6] Also record sleep start for a task group
Posted by Glauber Costa on Wed, 30 May 2012 09:48:36 GMT
View Forum Message <> Reply to Message

When we're dealing with a task group, instead of a task, also record
the start of its sleep time. Since the test agains TASK_UNINTERRUPTIBLE
does not really make sense and lack an obvious analogous, we always
record it as sleep_start, never block_start.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/fair.c | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index c26fe38..d932559 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -1182,7 +1182,8 @@ 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
  }

--
1.7.10.2

Subject: [PATCH v3 6/6] expose per-taskgroup schedstats in cgroup
Posted by Glauber Costa on Wed, 30 May 2012 09:48:37 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>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/core.c | 114 ++
 kernel/sched/fair.c | 24 ++++++++++
 kernel/sched/sched.h | 2 +
 3 files changed, 140 insertions(+)

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index db4f2c3..9c344d3 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -7969,6 +7969,107 @@ 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
+
+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;
+}
+
+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
```

```
  {
@@ -7976,6 +8077,19 @@ 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 d932559..7145c59 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:
  */
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index 76f6839..3d8ba03 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 {
--
1.7.10.2
```

Subject: Re: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Peter Zijlstra on Wed, 30 May 2012 10:29:19 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
> For simetry with the cfq tasks, measure exec_clock for the rt
> sched entities (rt_se).

Symmetry methinks.. anyway, where is the symmetry?, fair.c:update_curr()
doesn't do the for_each_sched_entity() thing.

> 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>
> CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
> CC: Paul Turner <pjt@google.com>
> ---
> kernel/sched/rt.c | 5 +++++
> kernel/sched/sched.h | 1 +
> 2 files changed, 6 insertions(+)
>

> diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
> index c5565c3..30ee4e2 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;

See, this just makes me sad.. you now have a double
for_each_sched_rt_entity() loop.

Subject: Re: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Glauber Costa on Wed, 30 May 2012 10:32:07 GMT
View Forum Message <> Reply to Message

On 05/30/2012 02:29 PM, Peter Zijlstra wrote:
> On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
>> For simetry with the cfq tasks, measure exec_clock for the rt
>> sched entities (rt_se).
>
> Symmetry methinks..

=p bad me
> anyway, where is the symmetry?, fair.c:update_curr()
> doesn't do the for_each_sched_entity() thing.

It does implicitly, because fair.c:update_curr() is called from
within enqueue_task(), that is called for_each_sched_entity in
enqueue_task_fair().

>
>> 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>
>> CC: Peter Zijlstra<a.p.zijlstra@chello.nl>
>> CC: Paul Turner<pjt@google.com>
>> ---

>> kernel/sched/rt.c | 5 +++++
>> kernel/sched/sched.h | 1 +
>> 2 files changed, 6 insertions(+)
>>
>> diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
>> index c5565c3..30ee4e2 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;
>
> See, this just makes me sad.. you now have a double
> for_each_sched_rt_entity() loop.

The way I read the rt.c code, it it is called from enqueue_task_rt only
once.

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Peter Zijlstra on Wed, 30 May 2012 10:34:01 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:

> +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;
> +}

Uhm, hotplug anyone?

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 10:34:08 GMT
View Forum Message <> Reply to Message

On 05/30/2012 02:34 PM, Peter Zijlstra wrote:
> On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
>
>> +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;
>> +}
>
> Uhm, hotplug anyone?
What's with hotplug ?

If you mean we should accumulate that on hotplug, I don't see why. We
certainly don't do that for the tick-based counters. Or do you mean I am
missing a get_online_cpus() lock ?

humm, I don't see it being taken on other loops like that

Subject: Re: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Peter Zijlstra on Wed, 30 May 2012 10:42:35 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 14:32 +0400, Glauber Costa wrote:
> >> +   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;
> >
> > See, this just makes me sad.. you now have a double
> > for_each_sched_rt_entity() loop.
>
> The way I read the rt.c code, it it is called from enqueue_task_rt only
> once.

Ah, what I meant was, right after that !rt_bandwidth_enabled() muck we
do another for_each_sched_rt_entity() walk.

Subject: Re: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Glauber Costa on Wed, 30 May 2012 10:42:35 GMT
View Forum Message <> Reply to Message

On 05/30/2012 02:42 PM, Peter Zijlstra wrote:
> On Wed, 2012-05-30 at 14:32 +0400, Glauber Costa wrote:
>>>> +   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;
>>>
>>> See, this just makes me sad.. you now have a double
>>> for_each_sched_rt_entity() loop.
>>
>> The way I read the rt.c code, it it is called from enqueue_task_rt only
>> once.
>
> Ah, what I meant was, right after that !rt_bandwidth_enabled() muck we
> do another for_each_sched_rt_entity() walk.
I guess I can fold it there...

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Peter Zijlstra on Wed, 30 May 2012 10:43:48 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 14:34 +0400, Glauber Costa wrote:
> On 05/30/2012 02:34 PM, Peter Zijlstra wrote:
> > On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
> >
> >> +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;
> >> +}
> >
> > Uhm, hotplug anyone?
> What's with hotplug ?

Who's to say all cpus are online at this point? If you don't want to
make the files come and go with hotplug, one should use
for_each_possible_cpu().

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 10:44:48 GMT
View Forum Message <> Reply to Message

On 05/30/2012 02:43 PM, Peter Zijlstra wrote:
> On Wed, 2012-05-30 at 14:34 +0400, Glauber Costa wrote:
>> On 05/30/2012 02:34 PM, Peter Zijlstra wrote:
>>> On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
>>>
>>>> +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;
>>>> +}
>>>
>>> Uhm, hotplug anyone?
>> What's with hotplug ?
>
>
> Who's to say all cpus are online at this point? If you don't want to
> make the files come and go with hotplug, one should use
> for_each_possible_cpu().
>
I don't oppose that.
But I don't really parse what you mean by "make the files go away". We
have just one file, with multiple entries. The entries are in key:value
form, so I don't see a huge problem in having some entries disappearing.

As a matter of fact, this is exactly what happens in /proc/stat when you
offline a cpu: the correspondent line will stop showing up.

Subject: Re: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Paul Turner on Wed, 30 May 2012 11:00:37 GMT
View Forum Message <> Reply to Message

On Wed, May 30, 2012 at 3:42 AM, Glauber Costa <glommer@parallels.com> wrote:
> On 05/30/2012 02:42 PM, Peter Zijlstra wrote:
>>
>> On Wed, 2012-05-30 at 14:32 +0400, Glauber Costa wrote:
>>>>>
>>>>> +   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;
>>>>
>>>>
>>>> See, this just makes me sad.. you now have a double

>>>> for_each_sched_rt_entity() loop.
>>>
>>>
>>> The way I read the rt.c code, it it is called from enqueue_task_rt only
>>> once.
>>
>>
>> Ah, what I meant was, right after that !rt_bandwidth_enabled() muck we
>> do another for_each_sched_rt_entity() walk.
>
> I guess I can fold it there...
>

Does this even need to be hierarchical? While it's natural for it to be in the CFS case, it feels forced here.

You could instead make this rt_rq->local_exec_clock charging only to the parenting rt_rq and post-aggregate when you want to report. The only thing you'd need to be careful of is also accounting children somewhere on the parent on destruction (reaped_exec_clock?).

Harking back to symmetry, local_exec_clock is also a potentially useful stat on the CFS side of things since it allows you to usefully disambiguate versus your children (common case where this is useful is calculating usage of threads in the root cgroup); so it wouldn't need to be unique to rt_rq.

Subject: Re: [PATCH v3 4/6] add a new scheduler hook for context switch
Posted by Peter Zijlstra on Wed, 30 May 2012 11:20:51 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
> To be able to count the number of switches per-cgroup, and merging
> Paul's hint that it would be better to do it in fair.c and rt.c,
> I am introducing a new write-side walk through a new scheduler hook,
> called at every context switch away from a task (prev).
>
> Read-side is greatly simplified, and as I'll show, the performance impact
> does not seem huge. First, aside from the function call, this walk is
> O(depth), which is not likely to be huge (if it is, the performance
> impact is indeed bad - but one can argue this is a good punishment)
>
> Also, this walk is likely to be cache-hot, since it is at most the very
> same loop done by put_prev_task, except it loops depth - 1 instead of depth
> times. This is specially important not to hurt tasks in the root cgroup,
> that will pay just a branch.

/me cries a little.. I was hoping to fix put_prev_task.. see:

https://lkml.org/lkml/2012/2/16/487

(I've actually got a 4 patch split out of that if anybody cares)

Its just one of those things stuck behind the -ENOTIME tree :/

The plan is to 'merge' put_prev_task and pick_next_task into one and
avoid a lot of the up-down walking.

You just added a constraint for always having to walk the entire thing
up -- cgroups is too damn expensive already, we should be trimming this
nonsense not bloating it.

> However, put_prev_task is called many times from multiple places,
> and the logic to differentiate a context switch from another kind of
> put would make a mess out of it.

I'm hoping the fold of put_prev in pick_next as per that patch I
referenced could help some, but the cross class switch makes that messy
still :/

Reducing the indirect calls is good, adding them is bad.. which makes me
the worst offender I'm afraid.

> On a 4-way x86_64, hackbench -pipe 1 process 4000 shows the following results:
> - units are seconds to complete the whole benchmark
> - percentual stdev for easier assesment
>
> Task sitting in the root cgroup:
> Without patchset: 4.857700 (0.69 %)
> With patchset: 4.863700 (0.63 %)
> Difference : 0.12 %

Just increase the repeat count :-)

$ perf stat -e cycles --repeat 100 perf bench sched messaging -p -g 100

```
    48,826,146,710 cycles  #    2.470 GHz  ( +-  0.17% )
      2.149005270 seconds time elapsed    ( +-  0.12% )
```

Anyway, a few nits on the below patch..

> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 4c1d7e9..db4f2c3 100644

> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -1894,6 +1894,14 @@ fire_sched_out_preempt_notifiers(struct task_struct *curr,
>
> #endif /* CONFIG_PREEMPT_NOTIFIERS */
>
> +static void

inline

> sched_class_context_switch(struct rq *rq, struct task_struct *prev)
> +{
> +#if defined(CONFIG_FAIR_GROUP_SCHED) || defined(CONFIG_RT_GROUP_SCHED)
> + if (prev->sched_class->context_switch)
> + prev->sched_class->context_switch(rq, prev);
> +#endif
> +}
> +

> diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
> index 940e6d1..c26fe38 100644
> --- a/kernel/sched/fair.c
> +++ b/kernel/sched/fair.c
> @@ -2993,6 +2993,20 @@ static struct task_struct *pick_next_task_fair(struct rq *rq)
>   return p;
> }
>
> +static void context_switch_fair(struct rq *rq, struct task_struct *p)
> +{
> +#ifdef CONFIG_FAIR_GROUP_SCHED
> + struct cfs_rq *cfs_rq;
> + struct sched_entity *se = &p->se;
> +
> + while (se->parent) {
> + se = se->parent;
> + cfs_rq = group_cfs_rq(se);
> + cfs_rq->nr_switches++;
> + }
> +#endif
> +}
> +

Put the whole function inside an existing #ifdef block of the right
kind.

> /*
>   * Account for a descheduled task:
>   */

> @@ -5255,6 +5269,7 @@ const struct sched_class fair_sched_class = {
> .check_preempt_curr = check_preempt_wakeup,
>
> .pick_next_task = pick_next_task_fair,
> + .context_switch = context_switch_fair,

Put the #ifdeffery here, so that the method is NULL when !FAIR_GROUP,
saves an indirect nop call for some weird .configs.

> .put_prev_task = put_prev_task_fair,
>
> #ifdef CONFIG_SMP

idem for sched/rt.c

Subject: Re: [PATCH v3 6/6] expose per-taskgroup schedstats in cgroup
Posted by Peter Zijlstra on Wed, 30 May 2012 11:22:47 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 13:48 +0400, Glauber Costa wrote:
> +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;

You just violated 2 coding style rules in one go :-)

If one side of the if-else has braces, the other side should have too.
If a block is multi-line (regardless of multi-stmt) it should have
braces.

/me hands you a bucket full of {}.

> +	return val;
> +}

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

On Wed, May 30, 2012 at 2:48 AM, Glauber Costa <glommer@parallels.com> wrote:
> The cpuacct cgroup already exposes user and system numbers in a per-cgroup
> fashion. But they are a summation along the whole group, not a per-cpu figure.
> Also, they are coarse-grained version of the stats usually shown at places
> like /proc/stat.
>
> I want to have enough cgroup data to emulate the /proc/stat interface. To
> achieve that, I am creating a new file "stat_percpu" that displays the
> fine-grained per-cpu data. The original data is left alone.
>
> The format of this file resembles the one found in the usual cgroup's stat
> files. But of course, the fields will be repeated, one per cpu, and prefixed
> with the cpu number.
>
> Therefore, we'll have something like:
>
> cpu0.user X
> cpu0.system Y
> ...
> cpu1.user X1
> cpu1.system Y1
> ...
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
> CC: Paul Turner <pjt@google.com>
> ---
> kernel/sched/core.c | 33 +++++++++++++++++++++++++++++++++
> 1 file changed, 33 insertions(+)
>
> diff --git a/kernel/sched/core.c b/kernel/sched/core.c
> index 220d416..4c1d7e9 100644
> --- a/kernel/sched/core.c
> +++ b/kernel/sched/core.c
> @@ -8178,6 +8178,35 @@ static int cpuacct_stats_show(struct cgroup *cgrp, struct cftype *cft,
> 	return 0;
> }
>

```
> +static inline void do_fill_cb(struct cgroup_map_cb *cb, struct cpuacct *ca,
> +                              char *str, int cpu, int index)
> +{
> +       char name[24];
> +       struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
> +
> +       snprintf(name, sizeof(name), "cpu%d.%s", cpu, str);
> +       cb->fill(cb, name, cputime64_to_clock_t(kcpustat->cpustat[index]));
> +}
> +
> +static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
> +                                    struct cgroup_map_cb *cb)
> +{
> +       struct cpuacct *ca = cgroup_ca(cgrp);
> +       int cpu;
> +
> +       for_each_online_cpu(cpu) {
> +               do_fill_cb(cb, ca, "user", cpu, CPUTIME_USER);
> +               do_fill_cb(cb, ca, "nice", cpu, CPUTIME_NICE);
> +               do_fill_cb(cb, ca, "system", cpu, CPUTIME_SYSTEM);
> +               do_fill_cb(cb, ca, "irq", cpu, CPUTIME_IRQ);
> +               do_fill_cb(cb, ca, "softirq", cpu, CPUTIME_SOFTIRQ);
> +               do_fill_cb(cb, ca, "guest", cpu, CPUTIME_GUEST);
> +               do_fill_cb(cb, ca, "guest_nice", cpu, CPUTIME_GUEST_NICE);
> +       }
> +
```

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

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

```
> +       return 0;
> +}
> +
>  static struct cftype files[] = {
>       {
>               .name = "usage",
> @@ -8192,6 +8221,10 @@ static struct cftype files[] = {
>               .name = "stat",
```

```
>                .read_map = cpuacct_stats_show,
>        },
> +      {
> +              .name = "stat_percpu",
> +              .read_map = cpuacct_stats_percpu_show,
> +      },
>        { }     /* terminate */
>  };
>
> --
> 1.7.10.2
>
```

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Peter Zijlstra on Wed, 30 May 2012 11:24:32 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 14:44 +0400, Glauber Costa wrote:
> But I don't really parse what you mean by "make the files go away". We
> have just one file, with multiple entries. The entries are in key:value
> form, so I don't see a huge problem in having some entries disappearing.
>
> As a matter of fact, this is exactly what happens in /proc/stat when you
> offline a cpu: the correspondent line will stop showing up.

Oh, I thought you had a file per cpu.

If you have content per cpu, this all might become a problem with 4096
cpus, that's bound to overflow the page of output space?

Subject: Re: [PATCH v3 5/6] Also record sleep start for a task group
Posted by Paul Turner on Wed, 30 May 2012 11:35:13 GMT
View Forum Message <> Reply to Message

On Wed, May 30, 2012 at 2:48 AM, Glauber Costa <glommer@parallels.com> wrote:
> When we're dealing with a task group, instead of a task, also record
> the start of its sleep time. Since the test agains TASK_UNINTERRUPTIBLE
> does not really make sense and lack an obvious analogous, we always
> record it as sleep_start, never block_start.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
> CC: Paul Turner <pjt@google.com>
> ---
> kernel/sched/fair.c | 3 ++-

> 1 file changed, 2 insertions(+), 1 deletion(-)
>
> diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
> index c26fe38..d932559 100644
> --- a/kernel/sched/fair.c
> +++ b/kernel/sched/fair.c
> @@ -1182,7 +1182,8 @@ 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;

You can't sanely account sleep on a group entity.

Suppose you have 2 sleepers on 1 cpu: you account 1s/s of idle
Suppose you have 2 sleepers now on 2 cpus: you account 2s/s of idle

Furthermore, in the latter case when one wakes up you still continue
to accrue sleep time whereas in the former you don't.

Just don't report/collect this.

> #endif
>   }
>
> --
> 1.7.10.2
>

Subject: Re: [PATCH v3 4/6] add a new scheduler hook for context switch
Posted by Peter Zijlstra on Wed, 30 May 2012 11:40:51 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 13:20 +0200, Peter Zijlstra wrote:
>
> $ perf stat -e cycles --repeat 100 perf bench sched messaging -p -g 100
>
>   48,826,146,710 cycles #   2.470 GHz (+- 0.17%)
>   2.149005270 seconds time elapsed   (+- 0.12%)

A repeat count of 250 dropped it down to:

       2.144582157 seconds time elapsed   (+- 0.08%)

Also, if you're poking at the context switch path, something like:

$ taskset 1 perf stat -e cycles --repeat 10 perf bench sched pipe -l 100000

gives a good number and is usually more stable than hackbench.

Subject: Re: [PATCH v3 4/6] add a new scheduler hook for context switch
Posted by Glauber Costa on Wed, 30 May 2012 12:07:57 GMT
View Forum Message <> Reply to Message

On 05/30/2012 03:20 PM, Peter Zijlstra wrote:
> /me cries a little.. I was hoping to fix put_prev_task.. see:
>
> https://lkml.org/lkml/2012/2/16/487
>
> (I've actually got a 4 patch split out of that if anybody cares)
>
> Its just one of those things stuck behind the -ENOTIME tree :/
>
> The plan is to 'merge' put_prev_task and pick_next_task into one and
> avoid a lot of the up-down walking.
>
> You just added a constraint for always having to walk the entire thing
> up -- cgroups is too damn expensive already, we should be trimming this
> nonsense not bloating it.

if they are merged, then I don't need a new hook.

Do you have plans to do it? I'd be happy to continue the work if you
lack the time, since I am directly interested in the functionality.

Subject: Re: [PATCH v3 4/6] add a new scheduler hook for context switch
Posted by Glauber Costa on Wed, 30 May 2012 12:08:24 GMT
View Forum Message <> Reply to Message

On 05/30/2012 03:40 PM, Peter Zijlstra wrote:
> On Wed, 2012-05-30 at 13:20 +0200, Peter Zijlstra wrote:
>>
>> $ perf stat -e cycles --repeat 100 perf bench sched messaging -p -g 100
>>
>> 48,826,146,710 cycles # 2.470 GHz (+- 0.17%)
>> 2.149005270 seconds time elapsed (+- 0.12%)
>
> A repeat count of 250 dropped it down to:
>

>   2.144582157 seconds time elapsed   (+- 0.08%)
>
> Also, if you're poking at the context switch path, something like:
>
> $ taskset 1 perf stat -e cycles --repeat 10 perf bench sched pipe -l 100000
>
> gives a good number and is usually more stable than hackbench.
thanks for the pointers

Subject: Re: [PATCH v3 1/6] measure exec_clock for rt sched entities
Posted by Glauber Costa on Wed, 30 May 2012 12:09:54 GMT
View Forum Message <> Reply to Message

On 05/30/2012 03:00 PM, Paul Turner wrote:
> Does this even need to be hierarchical? While it's natural for it to
> be in the CFS case, it feels forced here.
>
> You could instead make this rt_rq->local_exec_clock charging only to
> the parenting rt_rq and post-aggregate when you want to report. The
> only thing you'd need to be careful of is also accounting children
> somewhere on the parent on destruction (reaped_exec_clock?).
>
> Harking back to symmetry, local_exec_clock is also a potentially
> useful stat on the CFS side of things since it allows you to usefully
> disambiguate versus your children (common case where this is useful is
> calculating usage of threads in the root cgroup); so it wouldn't need
> to be unique to rt_rq.

I can try this approach.

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 12:20:15 GMT
View Forum Message <> Reply to Message

On 05/30/2012 03:24 PM, Paul Turner wrote:
>> +static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
>> > +   struct cgroup_map_cb *cb)
>> > +{
>> > +   struct cpuacct *ca = cgroup_ca(cgrp);
>> > +   int cpu;
>> > +
>> > +   for_each_online_cpu(cpu) {
>> > +   do_fill_cb(cb, ca, "user", cpu, CPUTIME_USER);
>> > +   do_fill_cb(cb, ca, "nice", cpu, CPUTIME_NICE);
>> > +   do_fill_cb(cb, ca, "system", cpu, CPUTIME_SYSTEM);

>> > + do_fill_cb(cb, ca, "irq", cpu, CPUTIME_IRQ);
>> > + do_fill_cb(cb, ca, "softirq", cpu, CPUTIME_SOFTIRQ);
>> > + do_fill_cb(cb, ca, "guest", cpu, CPUTIME_GUEST);
>> > + do_fill_cb(cb, ca, "guest_nice", cpu, CPUTIME_GUEST_NICE);
>> > + }
>> > +
> I don't know if there's much that can be trivially done about it but I
> suspect these are a bit of a memory allocation time-bomb on a many-CPU
> machine. The cgroup:seq_file mating (via read_map) treats everything
> as/one/ record. This means that seq_printf is going to end up
> eventually allocating a buffer that can fit_everything_ (as well as
> every power-of-2 on the way there). Adding insult to injury is that
> that the backing buffer is kmalloc() not vmalloc().
>
> 200+ bytes per-cpu above really is not unreasonable (46 bytes just for
> the text, plus a byte per base 10 digit we end up reporting), but that
> then leaves us looking at order-12/13 allocations just to print this
> thing when there are O(many) cpus.
>

And how's /proc/stat different ?
It will suffer from the very same problems, since it also have this very
same information (actually more, since I am skipping some), per-cpu.

Now, if you guys are okay with a file per-cpu, I can do it as well.
It pollutes the filesystem, but at least protects against the fact that
this is kmalloc-backed.

Subject: Re: [PATCH v3 5/6] Also record sleep start for a task group
Posted by Glauber Costa on Wed, 30 May 2012 12:24:40 GMT
View Forum Message <> Reply to Message

On 05/30/2012 03:35 PM, Paul Turner wrote:
> On Wed, May 30, 2012 at 2:48 AM, Glauber Costa<glommer@parallels.com> wrote:
>> When we're dealing with a task group, instead of a task, also record
>> the start of its sleep time. Since the test agains TASK_UNINTERRUPTIBLE
>> does not really make sense and lack an obvious analogous, we always
>> record it as sleep_start, never block_start.
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: Peter Zijlstra<a.p.zijlstra@chello.nl>
>> CC: Paul Turner<pjt@google.com>
>> ---
>> kernel/sched/fair.c | 3 ++-
>> 1 file changed, 2 insertions(+), 1 deletion(-)
>>
>> diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c

>> index c26fe38..d932559 100644
>> --- a/kernel/sched/fair.c
>> +++ b/kernel/sched/fair.c
>> @@ -1182,7 +1182,8 @@ 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;
>
> You can't sanely account sleep on a group entity.
>
> Suppose you have 2 sleepers on 1 cpu: you account 1s/s of idle
> Suppose you have 2 sleepers now on 2 cpus: you account 2s/s of idle
>
> Furthermore, in the latter case when one wakes up you still continue
> to accrue sleep time whereas in the former you don't.
>
> Just don't report/collect this.

sleep_start is not for iowait. This is for idle. And I know no other way
to collect idle time per cgroup, other than the time during which it was
out of the runqueue.

Now what you say about the sleepers don't make that much sense for idle
because this information is per-cpu as well.

When the se is being dequeued, it means none of its children is running
on that runqueue. That's idle.
>> #endif
>> }
>>
>> --
>> 1.7.10.2
>>

Subject: Re: [PATCH v3 5/6] Also record sleep start for a task group
Posted by Glauber Costa on Wed, 30 May 2012 12:44:09 GMT
View Forum Message <> Reply to Message

On 05/30/2012 04:44 PM, Peter Zijlstra wrote:
> On Wed, 2012-05-30 at 16:24 +0400, Glauber Costa wrote:
>> sleep_start is not for iowait. This is for idle. And I know no other way
>> to collect idle time per cgroup, other than the time during which it was
>> out of the runqueue.

>>
>> Now what you say about the sleepers don't make that much sense for idle
>> because this information is per-cpu as well.
>>
>> When the se is being dequeued, it means none of its children is running
>> on that runqueue. That's idle.
>
> But does that mean the cgroup is idle? Its impossible to re-construct
> the machine state from this per-cpu data if your definition of
> cgroup-idle is the time when _all_ cpus are idle.
>
It is idle for that runqueue, aka cpu. The cgroup itself is idle when
all cpus are idle. And yes, then you have 2s per sec of idle in a 2-way
system.

That's pretty much how a physical box works as well.

Subject: Re: [PATCH v3 5/6] Also record sleep start for a task group
Posted by Peter Zijlstra on Wed, 30 May 2012 12:44:37 GMT
View Forum Message <> Reply to Message

On Wed, 2012-05-30 at 16:24 +0400, Glauber Costa wrote:
> sleep_start is not for iowait. This is for idle. And I know no other way
> to collect idle time per cgroup, other than the time during which it was
> out of the runqueue.
>
> Now what you say about the sleepers don't make that much sense for idle
> because this information is per-cpu as well.
>
> When the se is being dequeued, it means none of its children is running
> on that runqueue. That's idle.

But does that mean the cgroup is idle? Its impossible to re-construct
the machine state from this per-cpu data if your definition of
cgroup-idle is the time when _all_ cpus are idle.

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Paul Turner on Wed, 30 May 2012 12:48:40 GMT
View Forum Message <> Reply to Message

On Wed, May 30, 2012 at 5:20 AM, Glauber Costa <glommer@parallels.com> wrote:
> On 05/30/2012 03:24 PM, Paul Turner wrote:
>>>
>>> +static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype
>>> *cft,

```
>>> >  +                               struct cgroup_map_cb *cb)
>>> >  +{
>>> >  +       struct cpuacct *ca = cgroup_ca(cgrp);
>>> >  +       int cpu;
>>> >  +
>>> >  +       for_each_online_cpu(cpu) {
>>> >  +               do_fill_cb(cb, ca, "user", cpu, CPUTIME_USER);
>>> >  +               do_fill_cb(cb, ca, "nice", cpu, CPUTIME_NICE);
>>> >  +               do_fill_cb(cb, ca, "system", cpu, CPUTIME_SYSTEM);
>>> >  +               do_fill_cb(cb, ca, "irq", cpu, CPUTIME_IRQ);
>>> >  +               do_fill_cb(cb, ca, "softirq", cpu, CPUTIME_SOFTIRQ);
>>> >  +               do_fill_cb(cb, ca, "guest", cpu, CPUTIME_GUEST);
>>> >  +               do_fill_cb(cb, ca, "guest_nice", cpu,
>>> > CPUTIME_GUEST_NICE);
>>> >  +       }
>>> >  +
>>
>> I don't know if there's much that can be trivially done about it but I
>> suspect these are a bit of a memory allocation time-bomb on a many-CPU
>> machine.  The cgroup:seq_file mating (via read_map) treats everything
>> as/one/  record.  This means that seq_printf is going to end up
>> eventually allocating a buffer that can fit_everything_  (as well as
>>
>> every power-of-2 on the way there).  Adding insult to injury is that
>> that the backing buffer is kmalloc() not vmalloc().
>>
>> 200+ bytes per-cpu above really is not unreasonable (46 bytes just for
>> the text, plus a byte per base 10 digit we end up reporting), but that
>> then  leaves us looking at order-12/13 allocations just to print this
>> thing when there are O(many) cpus.
>>
>
> And how's /proc/stat different ?
> It will suffer from the very same problems, since it also have this very
> same information (actually more, since I am skipping some), per-cpu.

So,

a) the information in /proc/stat is actually much denser since it's
"cpu VAL VAL VAL VAL" as opposed to "cpuX.FIELD VAL"
b) If it became a problem the /proc/stat case is actually fairly
trivially fixable by defining each cpu as a record and "everything
else" as a magic im-out-of-cpus value.

>
> Now, if you guys are okay with a file per-cpu, I can do it as well.
> It pollutes the filesystem, but at least protects against the fact that this
> is kmalloc-backed.
```

>

As I prefaced, I'm not sure there's much that can be trivially done
about it. This is really a fundamental limitation of how read_map()
works.

What we really need is a proper seq_file exposed through cftypes.

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 12:52:31 GMT
View Forum Message <> Reply to Message

On 05/30/2012 04:48 PM, Paul Turner wrote:
> a) the information in /proc/stat is actually much denser since it's
> "cpu VAL VAL VAL VAL" as opposed to "cpuX.FIELD VAL"

easily fixable here. Less descriptive, but we can use a header line
with the description much like how /proc/slabinfo does, and we still
have an extensible interface that is dense, at the same time.

> b) If it became a problem the /proc/stat case is actually fairly
> trivially fixable by defining each cpu as a record and "everything
> else" as a magic im-out-of-cpus value.
>
>> >
>> > Now, if you guys are okay with a file per-cpu, I can do it as well.
>> > It pollutes the filesystem, but at least protects against the fact that this
>> > is kmalloc-backed.
>> >
> As I prefaced, I'm not sure there's much that can be trivially done
> about it. This is really a fundamental limitation of how read_map()
> works.
>
> What we really need is a proper seq_file exposed through cftypes.
That can be done.

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 13:26:12 GMT
View Forum Message <> Reply to Message

On 05/30/2012 04:48 PM, Paul Turner wrote:
>> > Now, if you guys are okay with a file per-cpu, I can do it as well.
>> > It pollutes the filesystem, but at least protects against the fact that this
>> > is kmalloc-backed.
>> >

> As I prefaced, I'm not sure there's much that can be trivially done
> about it. This is really a fundamental limitation of how read_map()
> works.
>
> What we really need is a proper seq_file exposed through cftypes.

Tejun, would you be okay with an interface that exports somehow the raw seq_file in cgroups ?

This way we could call s_show for each cpu and get away with the memory usage problem, I presume

---

Subject: Re: [PATCH v3 3/6] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Wed, 30 May 2012 13:26:24 GMT
View Forum Message <> Reply to Message

On 05/30/2012 04:48 PM, Paul Turner wrote:
>> > Now, if you guys are okay with a file per-cpu, I can do it as well.
>> > It pollutes the filesystem, but at least protects against the fact that this
>> > is kmalloc-backed.
>> >
> As I prefaced, I'm not sure there's much that can be trivially done
> about it. This is really a fundamental limitation of how read_map()
> works.
>
> What we really need is a proper seq_file exposed through cftypes.

Tejun, would you be okay with an interface that exports somehow the raw seq_file in cgroups ?

This way we could call s_show for each cpu and get away with the memory usage problem, I presume

---