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
```