Subject: [PATCH 4/4] cpuacct.stat: re-use scheduler statistics for the root cgroup
Posted by Glauber Costa on Fri, 25 Nov 2011 01:33:26 GMT
View Forum Message <> Reply to Message

Right now, after we collect tick statistics for user and system and store them
in a well known location, we keep the same statistics again for cpuacct.
Since cpuacct is hierarchical, the numbers for the root cgroup should be
absolutely equal to the system-wide numbers.

So it would be better to just use it: this patch changes cpuacct accounting
in a way that the cpustat statistics are kept in a struct kernel_cpustat percpu
array. In the root cgroup case, we just point it to the main array. The rest of
the hierarchy walk can be totally disabled later with a static branch - but I am
not doing it here.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Paul Tuner <pjt@google.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
---
 kernel/sched.c | 118 ++++++++++++++++++++++++++++++++++++++----------------
 1 files changed, 72 insertions(+), 46 deletions(-)

```
diff --git a/kernel/sched.c b/kernel/sched.c
index 9b70305..9961817 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -1608,10 +1608,11 @@ struct cpuacct {
  struct cgroup_subsys_state css;
  /* cpuusage holds pointer to a u64-type object on every cpu */
  u64 __percpu *cpuusage;
- struct percpu_counter cpustat[CPUACCT_STAT_NSTATS];
+ struct kernel_cpustat __percpu *cpustat;
 };

 struct cgroup_subsys cpuacct_subsys;
+struct cpuacct root_cpuacct;

 /* return cpu accounting group corresponding to this container */
 static inline struct cpuacct *cgroup_ca(struct cgroup *cgrp)
@@ -1635,14 +1636,40 @@ static inline struct cpuacct *parent_ca(struct cpuacct *ca)
 }

 static void cpuacct_charge(struct task_struct *tsk, u64 cputime);
-static void cpuacct_update_stats(struct task_struct *tsk,
-  enum cpuacct_stat_index idx, cputime_t val);
 #else
 static inline void cpuacct_charge(struct task_struct *tsk, u64 cputime) {}
-static inline void cpuacct_update_stats(struct task_struct *tsk,
```

```
-  enum cpuacct_stat_index idx, cputime_t val) {}
 #endif

+static inline void task_group_account_field(struct task_struct *p,
+          u64 tmp, int index)
+{
+#ifdef CONFIG_CGROUP_CPUACCT
+ struct kernel_cpustat *kcpustat;
+ struct cpuacct *ca;
+#endif
+ /*
+  * Since all updates are sure to touch the root cgroup, we
+  * get ourselves ahead and touch it first. If the root cgroup
+  * is the only cgroup, then nothing else should be necessary.
+  *
+  */
+ __get_cpu_var(kernel_cpustat).cpustat[index] += tmp;
+
+#ifdef CONFIG_CGROUP_CPUACCT
+ if (unlikely(!cpuacct_subsys.active))
+  return;
+
+ rcu_read_lock();
+ ca = task_ca(p);
+ while (ca && (ca != &root_cpuacct)) {
+  kcpustat = this_cpu_ptr(ca->cpustat);
+  kcpustat->cpustat[index] += tmp;
+  ca = parent_ca(ca);
+ }
+ rcu_read_unlock();
+#endif
+}
+
 static inline void inc_cpu_load(struct rq *rq, unsigned long load)
 {
  update_load_add(&rq->load, load);
@@ -3921,7 +3948,7 @@ void account_user_time(struct task_struct *p, cputime_t cputime,
  index = (TASK_NICE(p) > 0) ? CPUTIME_NICE : CPUTIME_USER;
  cpustat[index] += tmp;

- cpuacct_update_stats(p, CPUACCT_STAT_USER, cputime);
+ task_group_account_field(p, index, cputime);
  /* Account for user time used */
  acct_update_integrals(p);
 }
@@ -3977,7 +4004,7 @@ void __account_system_time(struct task_struct *p, cputime_t cputime,

  /* Add system time to cpustat. */
```

```
  cpustat[index] += tmp;
- cpuacct_update_stats(p, CPUACCT_STAT_SYSTEM, cputime);
+ task_group_account_field(p, index, cputime);

  /* Account for system time used */
  acct_update_integrals(p);
@@ -8275,8 +8302,15 @@ void __init sched_init(void)
  list_add(&root_task_group.list, &task_groups);
  INIT_LIST_HEAD(&root_task_group.children);
  autogroup_init(&init_task);
+
 #endif /* CONFIG_CGROUP_SCHED */

+#ifdef CONFIG_CGROUP_CPUACCT
+ root_cpuacct.cpustat = &kernel_cpustat;
+ root_cpuacct.cpuusage = alloc_percpu(u64);
+ /* Too early, not expected to fail */
+ BUG_ON(!root_cpuacct.cpuusage);
+#endif
  for_each_possible_cpu(i) {
   struct rq *rq;

@@ -9537,9 +9571,12 @@ struct cgroup_subsys cpu_cgroup_subsys = {
 static struct cgroup_subsys_state *cpuacct_create(
  struct cgroup_subsys *ss, struct cgroup *cgrp)
 {
- struct cpuacct *ca = kzalloc(sizeof(*ca), GFP_KERNEL);
- int i;
+ struct cpuacct *ca;
+
+ if (!cgrp->parent)
+  return &root_cpuacct.css;

+ ca = kzalloc(sizeof(*ca), GFP_KERNEL);
  if (!ca)
   goto out;

@@ -9547,15 +9584,13 @@ static struct cgroup_subsys_state *cpuacct_create(
  if (!ca->cpuusage)
   goto out_free_ca;

- for (i = 0; i < CPUACCT_STAT_NSTATS; i++)
-  if (percpu_counter_init(&ca->cpustat[i], 0))
-   goto out_free_counters;
+ ca->cpustat = alloc_percpu(struct kernel_cpustat);
+ if (!ca->cpustat)
+  goto out_free_cpuusage;
```

```
  return &ca->css;

-out_free_counters:
- while (--i >= 0)
-  percpu_counter_destroy(&ca->cpustat[i]);
+out_free_cpuusage:
  free_percpu(ca->cpuusage);
 out_free_ca:
  kfree(ca);
@@ -9568,10 +9603,8 @@ static void
 cpuacct_destroy(struct cgroup_subsys *ss, struct cgroup *cgrp)
 {
  struct cpuacct *ca = cgroup_ca(cgrp);
- int i;

- for (i = 0; i < CPUACCT_STAT_NSTATS; i++)
-  percpu_counter_destroy(&ca->cpustat[i]);
+ free_percpu(ca->cpustat);
  free_percpu(ca->cpuusage);
  kfree(ca);
 }
@@ -9664,16 +9697,31 @@ static const char *cpuacct_stat_desc[] = {
 };

 static int cpuacct_stats_show(struct cgroup *cgrp, struct cftype *cft,
-  struct cgroup_map_cb *cb)
+         struct cgroup_map_cb *cb)
 {
  struct cpuacct *ca = cgroup_ca(cgrp);
- int i;
+ int cpu;
+ s64 val = 0;
+
+ for_each_online_cpu(cpu) {
+  struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
+  val += kcpustat->cpustat[CPUTIME_USER];
+  val += kcpustat->cpustat[CPUTIME_NICE];
+ }
+ val = cputime64_to_clock_t(val);
+ cb->fill(cb, cpuacct_stat_desc[CPUACCT_STAT_USER], val);

- for (i = 0; i < CPUACCT_STAT_NSTATS; i++) {
-  s64 val = percpu_counter_read(&ca->cpustat[i]);
-  val = cputime64_to_clock_t(val);
-  cb->fill(cb, cpuacct_stat_desc[i], val);
+ val = 0;
+ for_each_online_cpu(cpu) {
+  struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
```

```
+  val += kcpustat->cpustat[CPUTIME_SYSTEM];
+  val += kcpustat->cpustat[CPUTIME_IRQ];
+  val += kcpustat->cpustat[CPUTIME_SOFTIRQ];
  }
+
+ val = cputime64_to_clock_t(val);
+ cb->fill(cb, cpuacct_stat_desc[CPUACCT_STAT_SYSTEM], val);
+
  return 0;
 }

@@ -9742,28 +9790,6 @@ static void cpuacct_charge(struct task_struct *tsk, u64 cputime)
 #define CPUACCT_BATCH 0
 #endif

-/*
- * Charge the system/user time to the task's accounting group.
- */
-static void cpuacct_update_stats(struct task_struct *tsk,
-  enum cpuacct_stat_index idx, cputime_t val)
-{
- struct cpuacct *ca;
- int batch = CPUACCT_BATCH;
-
- if (unlikely(!cpuacct_subsys.active))
-  return;
-
- rcu_read_lock();
- ca = task_ca(tsk);
-
- do {
-  __percpu_counter_add(&ca->cpustat[idx], val, batch);
-  ca = parent_ca(ca);
- } while (ca);
- rcu_read_unlock();
-}
-
 struct cgroup_subsys cpuacct_subsys = {
  .name = "cpuacct",
  .create = cpuacct_create,
--
1.7.6.4
```