Subject: [PATCH] sched: cpu accounting controller (V2)
Posted by Srivatsa Vaddagiri on Fri, 30 Nov 2007 12:32:37 GMT
View Forum Message <> Reply to Message

On Fri, Nov 30, 2007 at 01:48:33AM +0530, Srivatsa Vaddagiri wrote:
> It is indeed an important todo. Right now we take a per-group global
> lock on every accounting update (which can be very frequent) and hence
> it is pretty bad.
>
> Ingo had expressed the need to reintroduce this patch asap and hence I resent
> it w/o attacking the scalability part. I will take a shot at scalability
> enhancements tomorrow and send it as a separate patch.

Here's V2 of the cpu acccounting controller patch, which makes
accounting scale better on SMP systems by splitting the usage counter to
be per-cpu.

---->

Commit cfb5285660aad4931b2ebbfa902ea48a37dfffa1 removed a useful feature for
us, which provided a cpu accounting resource controller. This feature would be
useful if someone wants to group tasks only for accounting purpose and doesnt
really want to exercise any control over their cpu consumption.

The patch below reintroduces the feature. It is based on Paul Menage's
original patch (Commit 62d0df64065e7c135d0002f069444fbdfc64768f), with
these differences:

- Removed load average information. I felt it needs more thought (esp
  to deal with SMP and virtualized platforms) and can be added for
  2.6.25 after more discussions.
- Convert group cpu usage to be nanosecond accurate (as rest of the cfs
  stats are) and invoke cpuacct_charge() from the respective scheduler
  classes
- Make accounting scalable on SMP systems by splitting the usage
  counter to be per-cpu
- Move the code from kernel/cpu_acct.c to kernel/sched.c (since the
  code is not big enough to warrant a new file and also this rightly
  needs to live inside the scheduler. Also things like accessing
  rq->lock while reading cpu usage becomes easier if the code lived in
  kernel/sched.c)

The patch also modifies the cpu controller not to provide the same accounting
information.

Signed-off-by: Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com>

```
---
 include/linux/cgroup_subsys.h |    7 +
 init/Kconfig                  |    7 +
 kernel/sched.c                |  155 ++++++++++++++++++++++++++++++++--------
 kernel/sched_fair.c           |    6 +
 kernel/sched_rt.c             |    1
 5 files changed, 150 insertions(+), 26 deletions(-)

Index: current/include/linux/cgroup_subsys.h
===================================================================
--- current.orig/include/linux/cgroup_subsys.h
+++ current/include/linux/cgroup_subsys.h
@@ -30,3 +30,10 @@ SUBSYS(cpu_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_CPUACCT
+SUBSYS(cpuacct)
+#endif
+
+/* */
+
Index: current/init/Kconfig
===================================================================
--- current.orig/init/Kconfig
+++ current/init/Kconfig
@@ -354,6 +354,13 @@ config FAIR_CGROUP_SCHED

 endchoice

+config CGROUP_CPUACCT
+ bool "Simple CPU accounting cgroup subsystem"
+ depends on CGROUPS
+ help
+   Provides a simple Resource Controller for monitoring the
+   total CPU consumed by the tasks in a cgroup
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -854,6 +854,12 @@ iter_move_one_task(struct rq *this_rq, i
      struct rq_iterator *iterator);
 #endif
```

```
+#ifdef CONFIG_CGROUP_CPUACCT
+static void cpuacct_charge(struct task_struct *tsk, u64 cputime);
+#else
+static inline void cpuacct_charge(struct task_struct *tsk, u64 cputime) {}
+#endif
+
 #include "sched_stats.h"
 #include "sched_idletask.c"
 #include "sched_fair.c"
@@ -7221,38 +7227,12 @@ static u64 cpu_shares_read_uint(struct c
  return (u64) tg->shares;
 }

-static u64 cpu_usage_read(struct cgroup *cgrp, struct cftype *cft)
-{
- struct task_group *tg = cgroup_tg(cgrp);
- unsigned long flags;
- u64 res = 0;
- int i;
-
- for_each_possible_cpu(i) {
-  /*
-   * Lock to prevent races with updating 64-bit counters
-   * on 32-bit arches.
-   */
-  spin_lock_irqsave(&cpu_rq(i)->lock, flags);
-  res += tg->se[i]->sum_exec_runtime;
-  spin_unlock_irqrestore(&cpu_rq(i)->lock, flags);
- }
- /* Convert from ns to ms */
- do_div(res, NSEC_PER_MSEC);
-
- return res;
-}
-
 static struct cftype cpu_files[] = {
  {
   .name = "shares",
   .read_uint = cpu_shares_read_uint,
   .write_uint = cpu_shares_write_uint,
  },
- {
-  .name = "usage",
-  .read_uint = cpu_usage_read,
- },
 };
```

```
 static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
@@ -7272,3 +7252,126 @@ struct cgroup_subsys cpu_cgroup_subsys =
 };

 #endif /* CONFIG_FAIR_CGROUP_SCHED */
+
+#ifdef CONFIG_CGROUP_CPUACCT
+
+/*
+ * CPU accounting code for task groups.
+ *
+ * Based on the work by Paul Menage (menage@google.com) and Balbir Singh
+ * (balbir@in.ibm.com).
+ */
+
+/* track cpu usage of a group of tasks */
+struct cpuacct {
+ struct cgroup_subsys_state css;
+ /* cpuusage holds pointer to a u64-type object on every cpu */
+ u64 *cpuusage;
+};
+
+struct cgroup_subsys cpuacct_subsys;
+
+/* return cpu accounting group corresponding to this container */
+static inline struct cpuacct *cgroup_ca(struct cgroup *cont)
+{
+ return container_of(cgroup_subsys_state(cont, cpuacct_subsys_id),
+       struct cpuacct, css);
+}
+
+/* return cpu accounting group to which this task belongs */
+static inline struct cpuacct *task_ca(struct task_struct *tsk)
+{
+ return container_of(task_subsys_state(tsk, cpuacct_subsys_id),
+       struct cpuacct, css);
+}
+
+/* create a new cpu accounting group */
+static struct cgroup_subsys_state *cpuacct_create(
+ struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct cpuacct *ca = kzalloc(sizeof(*ca), GFP_KERNEL);
+
+ if (!ca)
+  return ERR_PTR(-ENOMEM);
+
+ ca->cpuusage = alloc_percpu(u64);
```

```
+ if (!ca->cpuusage) {
+  kfree(ca);
+  return ERR_PTR(-ENOMEM);
+ }
+
+ return &ca->css;
+}
+
+/* destroy an existing cpu accounting group */
+static void cpuacct_destroy(struct cgroup_subsys *ss,
+       struct cgroup *cont)
+{
+ struct cpuacct *ca = cgroup_ca(cont);
+
+ free_percpu(ca->cpuusage);
+ kfree(ca);
+}
+
+/* return total cpu usage (in nanoseconds) of a group */
+static u64 cpuusage_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct cpuacct *ca = cgroup_ca(cont);
+ u64 totalcpuusage = 0;
+ int i;
+
+ for_each_possible_cpu(i) {
+  u64 *cpuusage = percpu_ptr(ca->cpuusage, i);
+
+  /*
+   * Take rq->lock to make 64-bit addition safe on 32-bit
+   * platforms.
+   */
+  spin_lock_irq(&cpu_rq(i)->lock);
+  totalcpuusage += *cpuusage;
+  spin_unlock_irq(&cpu_rq(i)->lock);
+ }
+
+ return totalcpuusage;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "usage",
+  .read_uint = cpuusage_read,
+ },
+};
+
+static int cpuacct_populate(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
+{
+ return cgroup_add_files(cont, ss, files, ARRAY_SIZE(files));
+}
+
+/*
+ * charge this task's execution time to its accounting group.
+ *
+ * called with rq->lock held.
+ */
+static void cpuacct_charge(struct task_struct *tsk, u64 cputime)
+{
+ struct cpuacct *ca;
+
+ if (!cpuacct_subsys.active)
+  return;
+
+ ca = task_ca(tsk);
+ if (ca) {
+  u64 *cpuusage = percpu_ptr(ca->cpuusage, task_cpu(tsk));
+
+  *cpuusage += cputime;
+ }
+}
+
+struct cgroup_subsys cpuacct_subsys = {
+ .name = "cpuacct",
+ .create = cpuacct_create,
+ .destroy = cpuacct_destroy,
+ .populate = cpuacct_populate,
+ .subsys_id = cpuacct_subsys_id,
+};
+#endif /* CONFIG_CGROUP_CPUACCT */
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -351,6 +351,12 @@ static void update_curr(struct cfs_rq *c

  __update_curr(cfs_rq, curr, delta_exec);
  curr->exec_start = now;
+
+ if (entity_is_task(curr)) {
+  struct task_struct *curtask = task_of(curr);
+
+  cpuacct_charge(curtask, delta_exec);
+ }
 }
```

```
 static inline void
Index: current/kernel/sched_rt.c
===================================================================
--- current.orig/kernel/sched_rt.c
+++ current/kernel/sched_rt.c
@@ -23,6 +23,7 @@ static void update_curr_rt(struct rq *rq

  curr->se.sum_exec_runtime += delta_exec;
  curr->se.exec_start = rq->clock;
+ cpuacct_charge(curr, delta_exec);
 }

 static void enqueue_task_rt(struct rq *rq, struct task_struct *p, int wakeup)
--
Regards,
vatsa
```

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