Subject: Re: [ckrm-tech] [PATCH 3/6] containers: Add generic multi-subsystem API to containers
Posted by Balbir Singh on Sat, 20 Jan 2007 17:27:44 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 1/11/07, Balbir Singh <balbir@in.ibm.com> wrote:
>> to 0. To walk the hierarchy, I have no root now since I do not have
>> any task context. I was wondering if exporting the rootnode or providing
>> a function to export the rootnode of the mounter hierarchy will make
>> programming easier.
>
> Ah - I misunderstood what you were looking for before.

Here it is, a simple patch to keep track of percentage cpu load of a container. This patch depends on the add mount callbacks patch and another patch that fixes cpuacct for powerpc boxes (posted previously).

The patch attempts to add a percentage load calculation for each container. It also maintains an accumulated time counter, which accounts for the total cpu time taken by the container.

Compiled and tested on a 4 cpu powerpc box. Paul, please include this in your next series of patches for containers.

Signed-off-by: <balbir@in.ibm.com>

```
 include/linux/cpu_acct.h |    4 ++
 kernel/cpu_acct.c        |   90
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++----
 kernel/sched.c           |    7 ++-
 3 files changed, 93 insertions(+), 8 deletions(-)

diff -puN kernel/cpu_acct.c~cpu_acct_load_acct kernel/cpu_acct.c
--- linux-2.6.20-rc5/kernel/cpu_acct.c~cpu_acct_load_acct 2007-01-20
18:28:26.000000000 +0530
+++ linux-2.6.20-rc5-balbir/kernel/cpu_acct.c 2007-01-20 22:32:49.000000000
+0530
@@ -13,16 +13,22 @@
 #include <linux/module.h>
 #include <linux/container.h>
 #include <linux/fs.h>
+#include <linux/time.h>
 #include <asm/div64.h>

 struct cpuacct {
  struct container_subsys_state css;
```

```
   spinlock_t lock;
   cputime64_t time; // total time used by this class
+ cputime64_t accum_time; // total time used by this class
  };

  static struct container_subsys cpuacct_subsys;
  static struct container *root;
+static spinlock_t interval_lock;
+static cputime64_t interval_time;
+static unsigned long long timestamp;
+static unsigned long long interval;

  static inline struct cpuacct *container_ca(struct container *cont)
  {
@@ -41,6 +47,8 @@ static int cpuacct_create(struct contain
   if (!ca) return -ENOMEM;
   spin_lock_init(&ca->lock);
   cont->subsys[cpuacct_subsys.subsys_id] = &ca->css;
+ ca->time = cputime64_zero;
+ ca->accum_time = cputime64_zero;
   return 0;
  }

@@ -67,17 +75,35 @@ static ssize_t cpuusage_read(struct cont
         size_t nbytes, loff_t *ppos)
  {
   struct cpuacct *ca = container_ca(cont);
- cputime64_t time;
- unsigned long time_in_jiffies;
+ unsigned long long time;
+ unsigned long long accum_time;
+ unsigned long long interval_jiffies;
   char usagebuf[64];
   char *s = usagebuf;

   spin_lock_irq(&ca->lock);
- time = ca->time;
+ time = cputime64_to_jiffies64(ca->time);
+ accum_time = cputime64_to_jiffies64(ca->accum_time);
   spin_unlock_irq(&ca->lock);

- time_in_jiffies = cputime_to_jiffies(time);
- s += sprintf(s, "%llu\n", (unsigned long long) time_in_jiffies);
+ spin_lock_irq(&interval_lock);
+ interval_jiffies = cputime64_to_jiffies64(interval_time);
+ spin_unlock_irq(&interval_lock);
+
+ s += sprintf(s, "time %llu\n", time);
```

```
+ s += sprintf(s, "accumulated time %llu\n", accum_time);
+ s += sprintf(s, "time since interval %llu\n", interval_jiffies);
+
+ /*
+  * Calculate time in percentage
+  */
+ time *= 100;
+ if (interval_jiffies)
+  do_div(time, interval_jiffies);
+ else
+  time = 0;
+
+ s += sprintf(s, "load %llu\n", time);

   return simple_read_from_buffer(buf, nbytes, ppos, usagebuf, s - usagebuf);
  }
@@ -96,7 +122,6 @@ static int cpuacct_populate(struct conta

  void cpuacct_charge(struct task_struct *task, cputime_t cputime)
  {
-
   struct cpuacct *ca;
   unsigned long flags;

@@ -106,11 +131,60 @@ void cpuacct_charge(struct task_struct *
   if (ca) {
    spin_lock_irqsave(&ca->lock, flags);
    ca->time = cputime64_add(ca->time, cputime);
+  ca->accum_time = cputime64_add(ca->accum_time, cputime);
    spin_unlock_irqrestore(&ca->lock, flags);
   }
   rcu_read_unlock();
  }

+void cpuacct_uncharge(struct task_struct *task, cputime_t cputime)
+{
+ struct cpuacct *ca;
+ unsigned long flags;
+
+ if (cpuacct_subsys.subsys_id < 0 || !root) return;
+ rcu_read_lock();
+ ca = task_ca(task);
+ if (ca) {
+  spin_lock_irqsave(&ca->lock, flags);
+  ca->time = cputime64_sub(ca->time, cputime);
+  ca->accum_time = cputime64_sub(ca->accum_time, cputime);
+  spin_unlock_irqrestore(&ca->lock, flags);
+ }
```

```
+ rcu_read_unlock();
+}
+
+static void reset_ca_time(struct container *root)
+{
+ struct container *child;
+ struct cpuacct *ca;
+
+ if (root) {
+  ca = container_ca(root);
+  if (ca) {
+   spin_lock(&ca->lock);
+   ca->time = cputime64_zero;
+   spin_unlock(&ca->lock);
+  }
+  list_for_each_entry(child, &root->children, sibling)
+   reset_ca_time(child);
+ }
+}
+
+void cpuacct_update_time(cputime_t cputime)
+{
+ unsigned long flags;
+ unsigned long long timestamp_now = get_jiffies_64();
+ spin_lock_irqsave(&interval_lock, flags);
+ interval_time += cputime_to_cputime64(cputime);
+ if ((timestamp_now - timestamp) > interval) {
+  timestamp = timestamp_now;
+  reset_ca_time(root);
+  interval_time = cputime64_zero;
+ }
+ spin_unlock_irqrestore(&interval_lock, flags);
+}
+
  static struct container_subsys cpuacct_subsys = {
   .name = "cpuacct",
   .create = cpuacct_create,
@@ -127,6 +201,10 @@ int __init init_cpuacct(void)
   int id;

   root = NULL;
+ interval = 10 * HZ;
+ interval_time = cputime64_zero;
+ timestamp = get_jiffies_64();
+ spin_lock_init(&interval_lock);
   id = container_register_subsys(&cpuacct_subsys);
   return id < 0 ? id : 0;
  }
```

```
diff -puN kernel/sched.c~cpu_acct_load_acct kernel/sched.c
--- linux-2.6.20-rc5/kernel/sched.c~cpu_acct_load_acct 2007-01-20
18:28:26.000000000 +0530
+++ linux-2.6.20-rc5-balbir/kernel/sched.c 2007-01-20 21:51:27.000000000 +0530
@@ -3078,7 +3078,7 @@ void account_user_time(struct task_struc
     */
   if (p != rq->idle)
    cpuacct_charge(p, cputime);
-
+ cpuacct_update_time(cputime);
   /* Add user time to cpustat. */
   tmp = cputime_to_cputime64(cputime);
   if (TASK_NICE(p) > 0)
@@ -3100,6 +3100,7 @@ void account_system_time(struct task_str
   struct rq *rq = this_rq();
   cputime64_t tmp;

+ cpuacct_update_time(cputime);
   p->stime = cputime_add(p->stime, cputime);

   /* Add system time to cpustat. */
@@ -3136,8 +3137,10 @@ void account_steal_time(struct task_stru
     cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
    else
     cpustat->idle = cputime64_add(cpustat->idle, tmp);
- } else
+ } else {
    cpustat->steal = cputime64_add(cpustat->steal, tmp);
+  cpuacct_uncharge(p, tmp);
+ }
  }

  static void task_running_tick(struct rq *rq, struct task_struct *p)
diff -puN include/linux/cpu_acct.h~cpu_acct_load_acct include/linux/cpu_acct.h
---  linux-2.6.20-rc5/include/linux/cpu_acct.h~cpu_acct_load_acct 2007-01-20
18:28:33.000000000 +0530
+++ linux-2.6.20-rc5-balbir/include/linux/cpu_acct.h 2007-01-20
21:51:43.000000000 +0530
@@ -7,8 +7,12 @@

  #ifdef CONFIG_CONTAINER_CPUACCT
  extern void cpuacct_charge(struct task_struct *, cputime_t cputime);
+extern void cpuacct_uncharge(struct task_struct *, cputime_t cputime);
+extern void cpuacct_update_time(cputime_t cputime);
  #else
  static void inline cpuacct_charge(struct task_struct *p, cputime_t cputime) {}
+static void inline cpuacct_uncharge(struct task_struct *p, cputime_t
cputime) {}
```

```
+static void inline cpuacct_update_time(cputime_t cputime) {}
 #endif

 #endif
```
–

--
Balbir Singh
Linux Technology Center
IBM, ISTL