
Subject: [PATCH 4/7] Containers (V8): Simple CPU accounting container subsystem

Posted by [Paul Menage](#) on Fri, 06 Apr 2007 23:32:25 GMT

[View Forum Message](#) <> [Reply to Message](#)

This example demonstrates how to use the generic container subsystem for a simple resource tracker that counts, for the processes in a container, the total CPU time used and the %CPU used in the last complete 10 second interval.

Portions contributed by Balbir Singh <balbir@in.ibm.com>

Signed-off-by: Paul Menage <menage@google.com>

```
include/linux/container_subsys.h | 6 +
include/linux/cpu_acct.h         | 14 ++
init/Kconfig                     | 7 +
kernel/Makefile                  | 1
kernel/cpu_acct.c                | 204 ++++++
kernel/sched.c                   | 14 ++
6 files changed, 243 insertions(+), 3 deletions(-)
```

Index: container-2.6.20-new/include/linux/cpu_acct.h

=====
--- /dev/null

+++ container-2.6.20-new/include/linux/cpu_acct.h

@@ -0,0 +1,14 @@

+

```
+#ifndef _LINUX_CPU_ACCT_H
+#define _LINUX_CPU_ACCT_H
```

+

```
+#include <linux/container.h>
```

```
+#include <asm/cputime.h>
```

+

```
+#ifdef CONFIG_CONTAINER_CPUACCT
```

```
+extern void cpuacct_charge(struct task_struct *, cputime_t cputime);
```

```
+#else
```

```
+static void inline cpuacct_charge(struct task_struct *p, cputime_t cputime) {}
```

```
+#endif
```

+

```
+#endif
```

Index: container-2.6.20-new/init/Kconfig

=====
--- container-2.6.20-new.orig/init/Kconfig

+++ container-2.6.20-new/init/Kconfig

```
@@ -278,6 +278,13 @@ config PROC_PID_CPUSET
 depends on CPUSETS
```

default y

```
+config CONTAINER_CPUACCT
+ bool "Simple CPU accounting container subsystem"
+ select CONTAINERS
+ help
+ Provides a simple Resource Controller for monitoring the
+ total CPU consumed by the tasks in a container
+
config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
Index: container-2.6.20-new/kernel/cpu_acct.c
```

```
=====
--- /dev/null
+++ container-2.6.20-new/kernel/cpu_acct.c
@@ -0,0 +1,204 @@
+/*
+ * kernel/cpu_acct.c - CPU accounting container subsystem
+ *
+ * Copyright (C) Google Inc, 2006
+ *
+ * Developed by Paul Menage (menage@google.com) and Balbir Singh
+ * (balbir@in.ibm.com)
+ *
+ */
+
+/*
+ * Container subsystem for reporting total CPU usage of tasks in a
+ * container, along with percentage load over a time interval
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+#include <asm/div64.h>
+
+struct cpuacct {
+ struct container_subsys_state css;
+ spinlock_t lock;
+ /* total time used by this class */
+ cputime64_t time;
+
+ /* time when next load calculation occurs */
+ u64 next_interval_check;
+
+ /* time used in current period */
+ cputime64_t current_interval_time;
```

```

+
+ /* time used in last period */
+ cputime64_t last_interval_time;
+};
+
+struct container_subsys cpuacct_subsys;
+
+static inline struct cpuacct *container_ca(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, cpuacct_subsys_id),
+ struct cpuacct, css);
+}
+
+static inline struct cpuacct *task_ca(struct task_struct *task)
+{
+ return container_ca(task_container(task, cpuacct_subsys_id));
+}
+
+#define INTERVAL (HZ * 10)
+
+static inline u64 next_interval_boundary(u64 now) {
+ /* calculate the next interval boundary beyond the
+ * current time */
+ do_div(now, INTERVAL);
+ return (now + 1) * INTERVAL;
+}
+
+static int cpuacct_create(struct container_subsys *ss, struct container *cont)
+{
+ struct cpuacct *ca = kzalloc(sizeof(*ca), GFP_KERNEL);
+ if (!ca)
+ return -ENOMEM;
+ spin_lock_init(&ca->lock);
+ ca->next_interval_check = next_interval_boundary(get_jiffies_64());
+ cont->subsys[cpuacct_subsys.subsys_id] = &ca->css;
+ return 0;
+}
+
+static void cpuacct_destroy(struct container_subsys *ss,
+ struct container *cont)
+{
+ kfree(container_ca(cont));
+}
+
+/* Lazily update the load calculation if necessary. Called with ca locked */
+static void cpuusage_update(struct cpuacct *ca)
+{
+ u64 now = get_jiffies_64();

```

```

+ /* If we're not due for an update, return */
+ if (ca->next_interval_check > now)
+ return;
+
+ if (ca->next_interval_check <= (now - INTERVAL)) {
+ /* If it's been more than an interval since the last
+ * check, then catch up - the last interval must have
+ * been zero load */
+ ca->last_interval_time = 0;
+ ca->next_interval_check = next_interval_boundary(now);
+ } else {
+ /* If a steal takes the last interval time negative,
+ * then we just ignore it */
+ if ((s64)ca->current_interval_time > 0) {
+ ca->last_interval_time = ca->current_interval_time;
+ } else {
+ ca->last_interval_time = 0;
+ }
+ ca->next_interval_check += INTERVAL;
+ }
+ ca->current_interval_time = 0;
+}
+
+static ssize_t cpuusage_read(struct container *cont,
+ struct cftype *cft,
+ struct file *file,
+ char __user *buf,
+ size_t nbytes, loff_t *ppos)
+{
+ struct cpuacct *ca = container_ca(cont);
+ u64 time;
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ spin_lock_irq(&ca->lock);
+ cpuusage_update(ca);
+ time = cputime64_to_jiffies64(ca->time);
+ spin_unlock_irq(&ca->lock);
+
+ /* Convert 64-bit jiffies to seconds */
+ time *= 1000;
+ do_div(time, HZ);
+ s += sprintf(s, "%llu", (unsigned long long) time);
+
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf, s - usagebuf);
+}
+
+static ssize_t load_read(struct container *cont,

```

```

+ struct cftype *cft,
+ struct file *file,
+ char __user *buf,
+ size_t nbytes, loff_t *ppos)
+{
+ struct cpuacct *ca = container_ca(cont);
+ u64 time;
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ /* Find the time used in the previous interval */
+ spin_lock_irq(&ca->lock);
+ cpuusage_update(ca);
+ time = cputime64_to_jiffies64(ca->last_interval_time);
+ spin_unlock_irq(&ca->lock);
+
+ /* Convert time to a percentage, to give the load in the
+ * previous period */
+ time *= 100;
+ do_div(time, INTERVAL);
+
+ s += sprintf(s, "%llu", (unsigned long long) time);
+
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf, s - usagebuf);
+}
+static struct cftype cft_usage = {
+ .name = "cpuacct.usage",
+ .read = cpuusage_read,
+};
+
+static struct cftype cft_load = {
+ .name = "cpuacct.load",
+ .read = load_read,
+};
+
+static int cpuacct_populate(struct container_subsys *ss,
+ struct container *cont)
+{
+ int err;
+
+ if ((err = container_add_file(cont, &cft_usage)))
+ return err;
+ if ((err = container_add_file(cont, &cft_load)))
+ return err;
+
+ return 0;
+}
+

```

```

+
+void cpuacct_charge(struct task_struct *task, cputime_t cputime)
+{
+
+ struct cpuacct *ca;
+ unsigned long flags;
+
+ if (!lcpuacct_subsys.active)
+ return;
+ rcu_read_lock();
+ ca = task_ca(task);
+ if (ca) {
+ spin_lock_irqsave(&ca->lock, flags);
+ cpuusage_update(ca);
+ ca->time = cputime64_add(ca->time, cputime);
+ ca->current_interval_time =
+ cputime64_add(ca->current_interval_time, cputime);
+ spin_unlock_irqrestore(&ca->lock, flags);
+ }
+ rcu_read_unlock();
+}
+
+struct container_subsys cpuacct_subsys = {
+ .name = "cpuacct",
+ .create = cpuacct_create,
+ .destroy = cpuacct_destroy,
+ .populate = cpuacct_populate,
+ .subsys_id = cpuacct_subsys_id,
+};

```

Index: container-2.6.20-new/kernel/Makefile

```

=====
--- container-2.6.20-new.orig/kernel/Makefile
+++ container-2.6.20-new/kernel/Makefile
@@ -37,6 +37,7 @@ obj-$(CONFIG_KEXEC) += kexec.o
obj-$(CONFIG_COMPAT) += compat.o
obj-$(CONFIG_CONTAINERS) += container.o
obj-$(CONFIG_CPUSETS) += cpuset.o
+obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
obj-$(CONFIG_IKCONFIG) += configs.o
obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
obj-$(CONFIG_AUDIT) += audit.o auditfilter.o

```

Index: container-2.6.20-new/kernel/sched.c

```

=====
--- container-2.6.20-new.orig/kernel/sched.c
+++ container-2.6.20-new/kernel/sched.c
@@ -52,6 +52,7 @@
#include <linux/tsacct_kern.h>
#include <linux/kprobes.h>

```

```

#include <linux/delayacct.h>
+#include <linux/cpu_acct.h>
#include <asm/tlb.h>

#include <asm/unistd.h>
@@ -3066,9 +3067,13 @@ void account_user_time(struct task_struct
{
    struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
    cputime64_t tmp;
+ struct rq *rq = this_rq();

    p->utime = cputime_add(p->utime, cputime);

+ if (p != rq->idle)
+   cpuacct_charge(p, cputime);
+
    /* Add user time to cpustat. */
    tmp = cputime_to_cputime64(cputime);
    if (TASK_NICE(p) > 0)
@@ -3098,9 +3103,10 @@ void account_system_time(struct task_struct
    cpustat->irq = cputime64_add(cpustat->irq, tmp);
    else if (softirq_count())
        cpustat->softirq = cputime64_add(cpustat->softirq, tmp);
- else if (p != rq->idle)
+ else if (p != rq->idle) {
    cpustat->system = cputime64_add(cpustat->system, tmp);
- else if (atomic_read(&rq->nr_iowait) > 0)
+   cpuacct_charge(p, cputime);
+ } else if (atomic_read(&rq->nr_iowait) > 0)
    cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
    else
        cpustat->idle = cputime64_add(cpustat->idle, tmp);
@@ -3125,8 +3131,10 @@ void account_steal_time(struct task_struct
    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_charge(p, -tmp);
+ }
}

static void task_running_tick(struct rq *rq, struct task_struct *p)
Index: container-2.6.20-new/include/linux/container_subsys.h
=====
--- container-2.6.20-new.orig/include/linux/container_subsys.h
+++ container-2.6.20-new/include/linux/container_subsys.h

```

```
@@ -11,4 +11,10 @@ SUBSYS(cpuset)
```

```
/* */
```

```
+#ifdef CONFIG_CONTAINER_CPUACCT
```

```
+SUBSYS(cpuacct)
```

```
+#endif
```

```
+
```

```
+/* */
```

```
+
```

```
/* */
```

```
--
```
