Subject: [PATCH] Cpu statistics accounting based on Paul Menage patches
Posted by xemul on Wed, 11 Apr 2007 14:58:38 GMT

View Forum Message <> Reply to Message

Provides a per-container statistics concerning the numbers of tasks
in various states, system and user times, etc. Patch is inspired
by Paul's example of the used CPU time accounting. Although this
patch is independent from Paul's example to make it possible playing
with them separately.

Based on Paul Menage containers v8.

One thing left is the idle time accounting. The problem in idle
time accounting is that the algorithm used to calculate global idle
time is useless. When system is loaded with tasks (and thus have
0% idle) some containers may be idle, i.e. no tasks run within a
container.

A known feature of this patch is that the numbers of tasks
shown for top container are not 100% true, as by the time the
top container is created system is already running. Probably
this statistics is simply not needed for top container.

```
--- ./include/linux/cpu_acct.h.taskstats 2007-04-11 15:07:44.000000000 +0400
+++ ./include/linux/cpu_acct.h 2007-04-11 16:50:22.000000000 +0400
@@ -0,0 +1,46 @@
+#ifndef _LINUX_CPU_ACCT_H
+#define _LINUX_CPU_ACCT_H
+
+#include <linux/container.h>
+
+#ifdef CONFIG_CPU_ACCT_CONTAINER
+void container_nr_running_inc(struct task_struct *);
+void container_nr_running_dec(struct task_struct *);
+
+void container_nr_unint_inc(struct task_struct *);
+void container_nr_unint_dec(struct task_struct *);
+
+void container_nr_sleeping_inc(struct task_struct *);
+void container_nr_sleeping_dec(struct task_struct *);
+
+void container_nr_stopped_inc(struct task_struct *);
+void container_nr_stopped_dec(struct task_struct *);
+
+void container_nr_iowait_inc(struct task_struct *);
+void container_nr_iowait_dec(struct task_struct *);
+
+void container_add_time_nice(struct task_struct *, cputime64_t);
```

```
+void container_add_time_user(struct task_struct *, cputime64_t);
+void container_add_time_system(struct task_struct *, cputime64_t);
+#else
+#define container_nr_running_inc(p) do { } while (0)
+#define container_nr_running_dec(p) do { } while (0)
+
+#define container_nr_unint_inc(p) do { } while (0)
+#define container_nr_unint_dec(p) do { } while (0)
+
+#define container_nr_sleeping_inc(p) do { } while (0)
+#define container_nr_sleeping_dec(p) do { } while (0)
+
+#define container_nr_stopped_inc(p) do { } while (0)
+#define container_nr_stopped_dec(p) do { } while (0)
+
+#define container_nr_iowait_inc(p) do { } while (0)
+#define container_nr_iowait_dec(p) do { } while (0)
+
+#define container_add_time_nice(p, time) do { } while (0)
+#define container_add_time_user(p, time) do { } while (0)
+#define container_add_time_system(p, time) do { } while (0)
+#endif
+
+#endif
--- ./include/linux/container_subsys.h.taskstats 2007-04-11 15:00:41.000000000 +0400
+++ ./include/linux/container_subsys.h 2007-04-11 16:06:53.000000000 +0400
@@ -9,6 +9,10 @@
 SUBSYS(cpuset)
 #endif

+#ifdef CONFIG_CPU_ACCT_CONTAINER
+SUBSYS(cpuacct)
+#endif
+
 /* */

 /* */
--- ./include/linux/sched.h.taskstats 2007-04-11 15:00:41.000000000 +0400
+++ ./include/linux/sched.h 2007-04-11 16:35:41.000000000 +0400
@@ -1052,6 +1052,9 @@ struct task_struct {
 #ifdef CONFIG_FAULT_INJECTION
  int make_it_fail;
 #endif
+#ifdef CONFIG_CPU_ACCT_CONTAINER
+ int iowait;
+#endif
 };
 static inline pid_t process_group(struct task_struct *tsk)
--- ./init/Kconfig.taskstats 2007-04-11 15:00:41.000000000 +0400
+++ ./init/Kconfig 2007-04-11 15:09:22.000000000 +0400
@@ -253,6 +253,13 @@ config CPUSETS

    Say N if unsure.

+config CPU_ACCT_CONTAINER
+ bool "CPU usage accounting container"
+ select CONTAINERS
+ help
+   Provides a per-container statistics concerning the numbers of tasks
+   in various states, system and user times, etc
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
--- ./kernel/cpu_acct.c.taskstats 2007-04-11 15:05:34.000000000 +0400
+++ ./kernel/cpu_acct.c 2007-04-11 18:25:12.000000000 +0400
@@ -0,0 +1,346 @@
+/*
+ * kernel/cpu_acct.c - CPU accounting container subsystem
+ *
+ * Developed by
+ *    Pavel Emelianov (xemul@openvz.org)
+ *
+ * Based on patch from
+ *    Paul Menage (menage@google.com) and Balbir Singh (balbir@in.ibm.com)
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+#include <linux/cpu_acct.h>
+
+enum {
+ CPUACCT_NR_RUNNING,
+ CPUACCT_NR_UNINT,
+ CPUACCT_NR_SLEEPING,
+ CPUACCT_NR_STOPPED,
+ CPUACCT_NR_IOWAIT,
+
+ CPUACCT_NR_MAX,
+};
+
+static char *cft_nr_names[] = {
+ "cpuacct_nr_running",
+ "cpuacct_nr_unint",
+ "cpuacct_nr_sleeping",
+ "cpuacct_nr_stopped",
+ "cpuacct_nr_iowait",
+};
+
+enum {
+ CPUACCT_TIME_NICE,
+ CPUACCT_TIME_USER,
+ CPUACCT_TIME_SYSTEM,
+
+ CPUACCT_TIME_MAX,
+};
+
+static char *cft_time_names[] = {
+ "cpuacct_time_nice",
+ "cpuacct_time_user",
+ "cpuacct_time_system",
+};
+
+struct cpuacct_counter {
+ long nrs[CPUACCT_NR_MAX];
+ cputime64_t times[CPUACCT_TIME_MAX];
+};
+
+struct cpuacct {
+ struct container_subsys_state css;
+ struct cpuacct_counter *counters;
+};
+
+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));
+}
+
+static int cpuacct_create(struct container_subsys *ss, struct container *cont)
+{
+ struct cpuacct *ca;
+
+ ca = kzalloc(sizeof(*ca), GFP_KERNEL);
+ if (ca == NULL)
+  goto out;
+
+ ca->counters = alloc_percpu(struct cpuacct_counter);
+ if (ca->counters == NULL)
+  goto out_free_ca;
+
+ cont->subsys[cpuacct_subsys.subsys_id] = &ca->css;
+ ca->css.container = cont;
+ return 0;
+
+out_free_ca:
+ kfree(ca);
+out:
+ return -ENOMEM;
+}
+
+static void cpuacct_destroy(struct container_subsys *ss,
+  struct container *cont)
+{
+ struct cpuacct *ca;
+
+ ca = container_ca(cont);
+ free_percpu(ca->counters);
+ kfree(ca);
+}
+
+static void cpuacct_attach(struct container_subsys *ss,
+  struct container *cont,
+  struct container *old_cont,
+  struct task_struct *p)
+{
+ int state, cpu;
+ struct cpuacct_counter *cnt_old, *cnt_new;
+
+ preempt_disable();
+ state = p->state;
+ cpu = smp_processor_id();
+ cnt_old = per_cpu_ptr(container_ca(old_cont)->counters, cpu);
+ cnt_new = per_cpu_ptr(container_ca(cont)->counters, cpu);
+
+ switch (state) {
+ case TASK_RUNNING:
+  cnt_old->nrs[CPUACCT_NR_RUNNING]--;
+  cnt_new->nrs[CPUACCT_NR_RUNNING]++;
+  break;
+ case TASK_UNINTERRUPTIBLE:
+  cnt_old->nrs[CPUACCT_NR_UNINT]--;
+  cnt_new->nrs[CPUACCT_NR_UNINT]++;
+  break;
+ case TASK_INTERRUPTIBLE:
+  cnt_old->nrs[CPUACCT_NR_SLEEPING]--;
+  cnt_new->nrs[CPUACCT_NR_SLEEPING]++;
+  break;
+ case TASK_STOPPED:
+  cnt_old->nrs[CPUACCT_NR_STOPPED]--;
+  cnt_new->nrs[CPUACCT_NR_STOPPED]++;
+  break;
+ }
+
+ if (p->iowait) {
+  cnt_old->nrs[CPUACCT_NR_IOWAIT]--;
+  cnt_new->nrs[CPUACCT_NR_IOWAIT]++;
+ }
+ preempt_enable();
+}
+
+static void cpuacct_exit(struct container_subsys *sub, struct task_struct *tsk)
+{
+ container_nr_running_dec(tsk);
+}
+
+static unsigned long cpuacct_sum_nr(struct cpuacct *ca, int nr_id)
+{
+ int cpu;
+ long sum;
+
+ sum = 0;
+ for_each_online_cpu(cpu)
+  sum += per_cpu_ptr(ca->counters, cpu)->nrs[nr_id];
+
+ return sum > 0 ? sum : 0;
+}
+
+static ssize_t cpuacct_read_nr(struct container *cont, struct cftype *cft,
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ char buf[64], *s;
+ unsigned long nr;
+
+ nr = cpuacct_sum_nr(container_ca(cont), cft->private);
+ s = buf;
+ s += sprintf(s, "%lu\n", nr);
+ return simple_read_from_buffer((void __user *)userbuf,
+   nbytes, ppos, buf, s - buf);
+}
```

```
+
+static unsigned long long cpuacct_sum_time(struct cpuacct *ca, int tid)
+{
+ int cpu;
+ unsigned long long sum;
+
+ sum = 0;
+ for_each_online_cpu(cpu)
+  sum += per_cpu_ptr(ca->counters, cpu)->times[tid];
+
+ return sum;
+}
+
+static ssize_t cpuacct_read_time(struct container *cont, struct cftype *cft,
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ char buf[64], *s;
+ unsigned long long nr;
+
+ nr = cpuacct_sum_time(container_ca(cont), cft->private);
+ s = buf;
+ s += sprintf(s, "%llu\n", nr);
+ return simple_read_from_buffer((void __user *)userbuf,
+   nbytes, ppos, buf, s - buf);
+}
+
+static struct cftype cft_nr_file[CPUACCT_NR_MAX];
+static struct cftype cft_time_file[CPUACCT_TIME_MAX];
+
+static int cpuacct_populate(struct container_subsys *ss,
+  struct container *cont)
+{
+ int i, err;
+
+ for (i = 0; i < CPUACCT_NR_MAX; i++) {
+  cft_nr_file[i].read = cpuacct_read_nr;
+  cft_nr_file[i].private = i;
+  strncpy(cft_nr_file[i].name, cft_nr_names[i],
+    MAX_CFTYPE_NAME);
+
+  err = container_add_file(cont, &cft_nr_file[i]);
+  if (err)
+   return err;
+ }
+
+ for (i = 0; i < CPUACCT_TIME_MAX; i++) {
+  cft_time_file[i].read = cpuacct_read_time;
+  cft_time_file[i].private = i;
+  strncpy(cft_time_file[i].name, cft_time_names[i],
+    MAX_CFTYPE_NAME);
+
+  err = container_add_file(cont, &cft_time_file[i]);
+  if (err)
+   return err;
+ }
+
+ return 0;
+}
+
+struct container_subsys cpuacct_subsys = {
+ .name = "cpuacct",
+ .create = cpuacct_create,
+ .destroy = cpuacct_destroy,
+ .populate = cpuacct_populate,
+ .attach = cpuacct_attach,
+ .exit = cpuacct_exit,
+ .subsys_id = cpuacct_subsys_id,
+};
+
+static void container_nr_mod(struct task_struct *tsk, int nr_id, int val)
+{
+ struct cpuacct *ca;
+ struct cpuacct_counter *cnt;
+
+ if (!cpuacct_subsys.active)
+  return;
+
+ rcu_read_lock();
+ ca = task_ca(tsk);
+ if (ca == NULL)
+  goto out;
+
+ cnt = per_cpu_ptr(ca->counters, smp_processor_id());
+ cnt->nrs[nr_id] += val;
+out:
+ rcu_read_unlock();
+}
+
+void container_nr_running_inc(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_RUNNING, 1);
+}
+
+void container_nr_running_dec(struct task_struct *p)
+{
```

```
+ container_nr_mod(p, CPUACCT_NR_RUNNING, -1);
+}
+
+void container_nr_unint_inc(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_UNINT, 1);
+}
+
+void container_nr_unint_dec(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_UNINT, -1);
+}
+
+void container_nr_sleeping_inc(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_SLEEPING, 1);
+}
+
+void container_nr_sleeping_dec(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_SLEEPING, -1);
+}
+
+void container_nr_stopped_inc(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_STOPPED, 1);
+}
+
+void container_nr_stopped_dec(struct task_struct *p)
+{
+ container_nr_mod(p, CPUACCT_NR_STOPPED, -1);
+}
+
+void container_nr_iowait_inc(struct task_struct *p)
+{
+ p->iowait = 1;
+ container_nr_mod(p, CPUACCT_NR_IOWAIT, 1);
+}
+
+void container_nr_iowait_dec(struct task_struct *p)
+{
+ p->iowait = 0;
+ container_nr_mod(p, CPUACCT_NR_IOWAIT, -1);
+}
+
+static void container_add_time(struct task_struct *p, int tid, cputime64_t time)
+{
+ struct cpuacct *ca;
+ struct cpuacct_counter *cnt;
+
+ if (!cpuacct_subsys.active)
+  return;
+
+ rcu_read_lock();
+ ca = task_ca(p);
+ if (ca == NULL)
+  goto out;
+
+ cnt = per_cpu_ptr(ca->counters, smp_processor_id());
+ cnt->times[tid] += time;
+out:
+ rcu_read_unlock();
+}
+
+void container_add_time_nice(struct task_struct *p, cputime64_t time)
+{
+ container_add_time(p, CPUACCT_TIME_NICE, time);
+}
+
+void container_add_time_user(struct task_struct *p, cputime64_t time)
+{
+ container_add_time(p, CPUACCT_TIME_USER, time);
+}
+
+void container_add_time_system(struct task_struct *p, cputime64_t time)
+{
+ container_add_time(p, CPUACCT_TIME_SYSTEM, time);
+}
--- ./kernel/sched.c.taskstats 2007-03-06 19:09:50.000000000 +0300
+++ ./kernel/sched.c 2007-04-11 16:50:19.000000000 +0400
@@ -56,6 +56,8 @@

 #include <asm/unistd.h>

+#include <linux/cpu_acct.h>
+
 /*
  * Convert user-nice values [ -20 ... 0 ... 19 ]
  * to static priority [ MAX_RT_PRIO..MAX_PRIO-1 ],
@@ -794,12 +796,14 @@ dec_raw_weighted_load(struct rq *rq, con
 static inline void inc_nr_running(struct task_struct *p, struct rq *rq)
 {
  rq->nr_running++;
+ container_nr_running_inc(p);
  inc_raw_weighted_load(rq, p);
 }
 static inline void dec_nr_running(struct task_struct *p, struct rq *rq)
 {
  rq->nr_running--;
+ container_nr_running_dec(p);
  dec_raw_weighted_load(rq, p);
 }

@@ -1004,6 +1008,13 @@ static void deactivate_task(struct task_
  dec_nr_running(p, rq);
  dequeue_task(p, p->array);
  p->array = NULL;
+
+ if (p->state == TASK_UNINTERRUPTIBLE)
+  container_nr_unint_inc(p);
+ else if (p->state == TASK_INTERRUPTIBLE)
+  container_nr_sleeping_inc(p);
+ else if (p->state == TASK_STOPPED)
+  container_nr_stopped_inc(p);
 }

 /*
@@ -1516,12 +1527,17 @@ out_activate:
 #endif /* CONFIG_SMP */
  if (old_state == TASK_UNINTERRUPTIBLE) {
   rq->nr_uninterruptible--;
+  container_nr_unint_dec(p);
   /*
    * Tasks on involuntary sleep don't earn
    * sleep_avg beyond just interactive state.
    */
   p->sleep_type = SLEEP_NONINTERACTIVE;
- } else
+ } else {
+  if (old_state == TASK_INTERRUPTIBLE)
+   container_nr_sleeping_dec(p);
+  else if (old_state == TASK_STOPPED)
+   container_nr_stopped_dec(p);

  /*
   * Tasks that have marked their sleep as noninteractive get
@@ -1530,6 +1546,7 @@ out_activate:
   */
   if (old_state & TASK_NONINTERACTIVE)
    p->sleep_type = SLEEP_NONINTERACTIVE;
+ }
```

```
  activate_task(p, rq, cpu == this_cpu);
@@ -3071,10 +3088,13 @@ void account_user_time(struct task_struc

  /* Add user time to cpustat. */
  tmp = cputime_to_cputime64(cputime);
- if (TASK_NICE(p) > 0)
+ if (TASK_NICE(p) > 0) {
   cpustat->nice = cputime64_add(cpustat->nice, tmp);
- else
+  container_add_time_nice(p, tmp);
+ } else {
   cpustat->user = cputime64_add(cpustat->user, tmp);
+  container_add_time_user(p, tmp);
+ }
 }

 /*
@@ -3091,6 +3111,7 @@ void account_system_time(struct task_str
  cputime64_t tmp;

  p->stime = cputime_add(p->stime, cputime);
+ container_add_time_system(p, cputime_to_cputime64(cputime));

  /* Add system time to cpustat. */
  tmp = cputime_to_cputime64(cputime);
@@ -4717,7 +4738,9 @@ void __sched io_schedule(void)

  delayacct_blkio_start();
  atomic_inc(&rq->nr_iowait);
+ container_nr_iowait_inc(current);
  schedule();
+ container_nr_iowait_dec(current);
  atomic_dec(&rq->nr_iowait);
  delayacct_blkio_end();
 }
@@ -4730,7 +4753,9 @@ long __sched io_schedule_timeout(long ti

  delayacct_blkio_start();
  atomic_inc(&rq->nr_iowait);
+ container_nr_iowait_inc(current);
  ret = schedule_timeout(timeout);
+ container_nr_iowait_dec(current);
  atomic_dec(&rq->nr_iowait);
  delayacct_blkio_end();
  return ret;
--- ./kernel/Makefile.taskstats 2007-04-11 15:00:41.000000000 +0400
+++ ./kernel/Makefile 2007-04-11 16:07:14.000000000 +0400
@@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
```

```
 obj-$(CONFIG_UTS_NS) += utsname.o
 obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
 obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
+obj-$(CONFIG_CPU_ACCT_CONTAINER) += cpu_acct.o

 ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
 # According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-frame-pointer is
```