Subject: [PATCH v2 0/3] cpuacct cgroup refactoring
Posted by Glauber Costa on Mon, 28 Nov 2011 16:45:16 GMT
View Forum Message <> Reply to Message

Hi,

This is the same series I've already sent before. Just sending again
after the conflict with the code that was in tip.git

Best regards,

Glauber Costa (3):
  Change cpustat fields to an array.
  Reuse cgroup's parent pointer
  cpuacct.stat: re-use scheduler statistics for the root cgroup

 arch/s390/appldata/appldata_os.c | 16 +-
 arch/x86/include/asm/i387.h | 2 +-
 drivers/cpufreq/cpufreq_conservative.c | 38 +++---
 drivers/cpufreq/cpufreq_ondemand.c | 38 +++---
 drivers/macintosh/rack-meter.c | 8 +-
 fs/proc/stat.c | 63 ++++-----
 fs/proc/uptime.c | 4 +-
 include/linux/kernel_stat.h | 36 +++--
 kernel/sched/core.c | 242 ++++++++++++++++++++----------
 kernel/sched/sched.h | 33 ++++-
 10 files changed, 248 insertions(+), 232 deletions(-)

--
1.7.6.4

Subject: [PATCH v2 1/3] Change cpustat fields to an array.
Posted by Glauber Costa on Mon, 28 Nov 2011 16:45:17 GMT
View Forum Message <> Reply to Message

This patch changes fields in cpustat from a structure, to an
u64 array. Math gets easier, and the code is more flexible.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Paul Tuner <pjt@google.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
---
 arch/s390/appldata/appldata_os.c | 16 +++---
 arch/x86/include/asm/i387.h | 2 +-
 drivers/cpufreq/cpufreq_conservative.c | 38 +++++++--------
 drivers/cpufreq/cpufreq_ondemand.c | 38 +++++++--------

```
 drivers/macintosh/rack-meter.c        |    8 ++--
 fs/proc/stat.c                        |   63 ++++++++++++++++++-------------
 fs/proc/uptime.c                      |    4 +-
 include/linux/kernel_stat.h           |   36 +++++++++------
 kernel/sched/core.c                   |   78 +++++++++++++++++++++++++++++++++++
 9 files changed, 142 insertions(+), 141 deletions(-)

diff --git a/arch/s390/appldata/appldata_os.c b/arch/s390/appldata/appldata_os.c
index 92f1cb7..4de031d 100644
--- a/arch/s390/appldata/appldata_os.c
+++ b/arch/s390/appldata/appldata_os.c
@@ -115,21 +115,21 @@ static void appldata_get_os_data(void *data)
  j = 0;
  for_each_online_cpu(i) {
   os_data->os_cpu[j].per_cpu_user =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.user);
+   cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_USER]);
   os_data->os_cpu[j].per_cpu_nice =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.nice);
+   cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_NICE]);
   os_data->os_cpu[j].per_cpu_system =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.system);
+    cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_SYSTEM]);
   os_data->os_cpu[j].per_cpu_idle =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.idle);
+   cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_IDLE]);
   os_data->os_cpu[j].per_cpu_irq =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.irq);
+   cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_IRQ]);
   os_data->os_cpu[j].per_cpu_softirq =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.softirq);
+    cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_SOFTIRQ]) ;
   os_data->os_cpu[j].per_cpu_iowait =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.iowait);
+    cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_IOWAIT]);
   os_data->os_cpu[j].per_cpu_steal =
-   cputime_to_jiffies(kstat_cpu(i).cpustat.steal);
+   cputime_to_jiffies(kcpustat_cpu(i).cpustat[CPUTIME_STEAL]);
   os_data->os_cpu[j].cpu_id = i;
   j++;
  }
diff --git a/arch/x86/include/asm/i387.h b/arch/x86/include/asm/i387.h
index c9e09ea..6919e93 100644
--- a/arch/x86/include/asm/i387.h
+++ b/arch/x86/include/asm/i387.h
@@ -218,7 +218,7 @@ static inline void fpu_fxsave(struct fpu *fpu)
 #ifdef CONFIG_SMP
 #define safe_address (__per_cpu_offset[0])
```

```
 #else
-#define safe_address (kstat_cpu(0).cpustat.user)
+#define safe_address (__get_cpu_var(kernel_cpustat).cpustat[CPUTIME_USER])
 #endif

 /*
diff --git a/drivers/cpufreq/cpufreq_conservative.c b/drivers/cpufreq/cpufreq_conservative.c
index c97b468..118bff7 100644
--- a/drivers/cpufreq/cpufreq_conservative.c
+++ b/drivers/cpufreq/cpufreq_conservative.c
@@ -95,27 +95,26 @@ static struct dbs_tuners {
  .freq_step = 5,
 };

-static inline cputime64_t get_cpu_idle_time_jiffy(unsigned int cpu,
-       cputime64_t *wall)
+static inline u64 get_cpu_idle_time_jiffy(unsigned int cpu, u64 *wall)
 {
- cputime64_t idle_time;
+ u64 idle_time;
  cputime64_t cur_wall_time;
- cputime64_t busy_time;
+ u64 busy_time;

  cur_wall_time = jiffies64_to_cputime64(get_jiffies_64());
- busy_time = cputime64_add(kstat_cpu(cpu).cpustat.user,
-   kstat_cpu(cpu).cpustat.system);
+ busy_time = kcpustat_cpu(cpu).cpustat[CPUTIME_USER] +
+       kcpustat_cpu(cpu).cpustat[CPUTIME_SYSTEM];

- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.irq);
- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.softirq);
- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.steal);
- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.nice);
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_IRQ];
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_SOFTIRQ];
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_STEAL];
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_NICE];

  idle_time = cputime64_sub(cur_wall_time, busy_time);
  if (wall)
-  *wall = (cputime64_t)jiffies_to_usecs(cur_wall_time);
+  *wall = jiffies_to_usecs(cur_wall_time);

- return (cputime64_t)jiffies_to_usecs(idle_time);
+ return jiffies_to_usecs(idle_time);
 }
```

```
 static inline cputime64_t get_cpu_idle_time(unsigned int cpu, cputime64_t *wall)
@@ -272,7 +271,7 @@ static ssize_t store_ignore_nice_load(struct kobject *a, struct attribute *b,
   dbs_info->prev_cpu_idle = get_cpu_idle_time(j,
       &dbs_info->prev_cpu_wall);
   if (dbs_tuners_ins.ignore_nice)
-   dbs_info->prev_cpu_nice = kstat_cpu(j).cpustat.nice;
+   dbs_info->prev_cpu_nice = kcpustat_cpu(j).cpustat[CPUTIME_NICE];
  }
  return count;
 }
@@ -362,11 +361,11 @@ static void dbs_check_cpu(struct cpu_dbs_info_s *this_dbs_info)
   j_dbs_info->prev_cpu_idle = cur_idle_time;

   if (dbs_tuners_ins.ignore_nice) {
-   cputime64_t cur_nice;
+   u64 cur_nice;
    unsigned long cur_nice_jiffies;

-   cur_nice = cputime64_sub(kstat_cpu(j).cpustat.nice,
-        j_dbs_info->prev_cpu_nice);
+   cur_nice = kcpustat_cpu(j).cpustat[CPUTIME_NICE] -
+        j_dbs_info->prev_cpu_nice;
    /*
     * Assumption: nice time between sampling periods will
     * be less than 2^32 jiffies for 32 bit sys
@@ -374,7 +373,7 @@ static void dbs_check_cpu(struct cpu_dbs_info_s *this_dbs_info)
    cur_nice_jiffies = (unsigned long)
      cputime64_to_jiffies64(cur_nice);

-   j_dbs_info->prev_cpu_nice = kstat_cpu(j).cpustat.nice;
+   j_dbs_info->prev_cpu_nice = kcpustat_cpu(j).cpustat[CPUTIME_NICE];
    idle_time += jiffies_to_usecs(cur_nice_jiffies);
   }

@@ -501,10 +500,9 @@ static int cpufreq_governor_dbs(struct cpufreq_policy *policy,

    j_dbs_info->prev_cpu_idle = get_cpu_idle_time(j,
       &j_dbs_info->prev_cpu_wall);
-   if (dbs_tuners_ins.ignore_nice) {
+   if (dbs_tuners_ins.ignore_nice)
     j_dbs_info->prev_cpu_nice =
-        kstat_cpu(j).cpustat.nice;
-   }
+        kcpustat_cpu(j).cpustat[CPUTIME_NICE];
   }
   this_dbs_info->down_skip = 0;
   this_dbs_info->requested_freq = policy->cur;
diff --git a/drivers/cpufreq/cpufreq_ondemand.c b/drivers/cpufreq/cpufreq_ondemand.c
```

```
index fa8af4e..f3d327c 100644
--- a/drivers/cpufreq/cpufreq_ondemand.c
+++ b/drivers/cpufreq/cpufreq_ondemand.c
@@ -119,27 +119,26 @@ static struct dbs_tuners {
  .powersave_bias = 0,
 };

-static inline cputime64_t get_cpu_idle_time_jiffy(unsigned int cpu,
-       cputime64_t *wall)
+static inline u64 get_cpu_idle_time_jiffy(unsigned int cpu, u64 *wall)
 {
- cputime64_t idle_time;
+ u64 idle_time;
  cputime64_t cur_wall_time;
- cputime64_t busy_time;
+ u64 busy_time;

  cur_wall_time = jiffies64_to_cputime64(get_jiffies_64());
- busy_time = cputime64_add(kstat_cpu(cpu).cpustat.user,
-   kstat_cpu(cpu).cpustat.system);
+ busy_time = kcpustat_cpu(cpu).cpustat[CPUTIME_USER] +
+       kcpustat_cpu(cpu).cpustat[CPUTIME_SYSTEM];

- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.irq);
- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.softirq);
- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.steal);
- busy_time = cputime64_add(busy_time, kstat_cpu(cpu).cpustat.nice);
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_IRQ];
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_SOFTIRQ];
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_STEAL];
+ busy_time += kcpustat_cpu(cpu).cpustat[CPUTIME_NICE];

  idle_time = cputime64_sub(cur_wall_time, busy_time);
  if (wall)
-  *wall = (cputime64_t)jiffies_to_usecs(cur_wall_time);
+  *wall = jiffies_to_usecs(cur_wall_time);

- return (cputime64_t)jiffies_to_usecs(idle_time);
+ return jiffies_to_usecs(idle_time);
 }

 static inline cputime64_t get_cpu_idle_time(unsigned int cpu, cputime64_t *wall)
@@ -345,7 +344,7 @@ static ssize_t store_ignore_nice_load(struct kobject *a, struct attribute *b,
    dbs_info->prev_cpu_idle = get_cpu_idle_time(j,
        &dbs_info->prev_cpu_wall);
    if (dbs_tuners_ins.ignore_nice)
-    dbs_info->prev_cpu_nice = kstat_cpu(j).cpustat.nice;
+    dbs_info->prev_cpu_nice = kcpustat_cpu(j).cpustat[CPUTIME_NICE];
```

```
 }
 return count;
@@ -455,11 +454,11 @@ static void dbs_check_cpu(struct cpu_dbs_info_s *this_dbs_info)
  j_dbs_info->prev_cpu_iowait = cur_iowait_time;

  if (dbs_tuners_ins.ignore_nice) {
-  cputime64_t cur_nice;
+  u64 cur_nice;
   unsigned long cur_nice_jiffies;

-  cur_nice = cputime64_sub(kstat_cpu(j).cpustat.nice,
-    j_dbs_info->prev_cpu_nice);
+  cur_nice = kcpustat_cpu(j).cpustat[CPUTIME_NICE] -
+    j_dbs_info->prev_cpu_nice;
   /*
    * Assumption: nice time between sampling periods will
    * be less than 2^32 jiffies for 32 bit sys
@@ -467,7 +466,7 @@ static void dbs_check_cpu(struct cpu_dbs_info_s *this_dbs_info)
   cur_nice_jiffies = (unsigned long)
    cputime64_to_jiffies64(cur_nice);

-  j_dbs_info->prev_cpu_nice = kstat_cpu(j).cpustat.nice;
+  j_dbs_info->prev_cpu_nice = kcpustat_cpu(j).cpustat[CPUTIME_NICE];
   idle_time += jiffies_to_usecs(cur_nice_jiffies);
  }

@@ -646,10 +645,9 @@ static int cpufreq_governor_dbs(struct cpufreq_policy *policy,

   j_dbs_info->prev_cpu_idle = get_cpu_idle_time(j,
     &j_dbs_info->prev_cpu_wall);
-  if (dbs_tuners_ins.ignore_nice) {
+  if (dbs_tuners_ins.ignore_nice)
    j_dbs_info->prev_cpu_nice =
-     kstat_cpu(j).cpustat.nice;
-  }
+     kcpustat_cpu(j).cpustat[CPUTIME_NICE];
  }
  this_dbs_info->cpu = cpu;
  this_dbs_info->rate_mult = 1;
diff --git a/drivers/macintosh/rack-meter.c b/drivers/macintosh/rack-meter.c
index 2637c13..66d7f1c7 100644
--- a/drivers/macintosh/rack-meter.c
+++ b/drivers/macintosh/rack-meter.c
@@ -81,13 +81,13 @@ static int rackmeter_ignore_nice;
 */
 static inline cputime64_t get_cpu_idle_time(unsigned int cpu)
 {
```

```
- cputime64_t retval;
+ u64 retval;

- retval = cputime64_add(kstat_cpu(cpu).cpustat.idle,
-   kstat_cpu(cpu).cpustat.iowait);
+ retval = kcpustat_cpu(cpu).cpustat[CPUTIME_IDLE] +
+    kcpustat_cpu(cpu).cpustat[CPUTIME_IOWAIT];

  if (rackmeter_ignore_nice)
-  retval = cputime64_add(retval, kstat_cpu(cpu).cpustat.nice);
+  retval += kcpustat_cpu(cpu).cpustat[CPUTIME_NICE];

  return retval;
 }
diff --git a/fs/proc/stat.c b/fs/proc/stat.c
index 42b274d..8a6ab66 100644
--- a/fs/proc/stat.c
+++ b/fs/proc/stat.c
@@ -22,29 +22,27 @@
 #define arch_idle_time(cpu) 0
 #endif

-static cputime64_t get_idle_time(int cpu)
+static u64 get_idle_time(int cpu)
 {
- u64 idle_time = get_cpu_idle_time_us(cpu, NULL);
- cputime64_t idle;
+ u64 idle, idle_time = get_cpu_idle_time_us(cpu, NULL);

  if (idle_time == -1ULL) {
   /* !NO_HZ so we can rely on cpustat.idle */
-  idle = kstat_cpu(cpu).cpustat.idle;
-  idle = cputime64_add(idle, arch_idle_time(cpu));
+  idle = kcpustat_cpu(cpu).cpustat[CPUTIME_IDLE];
+  idle += arch_idle_time(cpu);
  } else
   idle = usecs_to_cputime(idle_time);

  return idle;
 }

-static cputime64_t get_iowait_time(int cpu)
+static u64 get_iowait_time(int cpu)
 {
- u64 iowait_time = get_cpu_iowait_time_us(cpu, NULL);
- cputime64_t iowait;
+ u64 iowait, iowait_time = get_cpu_iowait_time_us(cpu, NULL);
```

```
  if (iowait_time == -1ULL)
   /* !NO_HZ so we can rely on cpustat.iowait */
-  iowait = kstat_cpu(cpu).cpustat.iowait;
+  iowait = kcpustat_cpu(cpu).cpustat[CPUTIME_IOWAIT];
  else
   iowait = usecs_to_cputime(iowait_time);

@@ -55,33 +53,30 @@ static int show_stat(struct seq_file *p, void *v)
 {
  int i, j;
  unsigned long jif;
- cputime64_t user, nice, system, idle, iowait, irq, softirq, steal;
- cputime64_t guest, guest_nice;
+ u64 user, nice, system, idle, iowait, irq, softirq, steal;
+ u64 guest, guest_nice;
  u64 sum = 0;
  u64 sum_softirq = 0;
  unsigned int per_softirq_sums[NR_SOFTIRQS] = {0};
  struct timespec boottime;

  user = nice = system = idle = iowait =
-  irq = softirq = steal = cputime64_zero;
- guest = guest_nice = cputime64_zero;
+  irq = softirq = steal = 0;
+ guest = guest_nice = 0;
  getboottime(&boottime);
  jif = boottime.tv_sec;

  for_each_possible_cpu(i) {
-  user = cputime64_add(user, kstat_cpu(i).cpustat.user);
-  nice = cputime64_add(nice, kstat_cpu(i).cpustat.nice);
-  system = cputime64_add(system, kstat_cpu(i).cpustat.system);
-  idle = cputime64_add(idle, get_idle_time(i));
-  iowait = cputime64_add(iowait, get_iowait_time(i));
-  irq = cputime64_add(irq, kstat_cpu(i).cpustat.irq);
-  softirq = cputime64_add(softirq, kstat_cpu(i).cpustat.softirq);
-  steal = cputime64_add(steal, kstat_cpu(i).cpustat.steal);
-  guest = cputime64_add(guest, kstat_cpu(i).cpustat.guest);
-  guest_nice = cputime64_add(guest_nice,
-   kstat_cpu(i).cpustat.guest_nice);
-  sum += kstat_cpu_irqs_sum(i);
-  sum += arch_irq_stat_cpu(i);
+  user += kcpustat_cpu(i).cpustat[CPUTIME_USER];
+  nice += kcpustat_cpu(i).cpustat[CPUTIME_NICE];
+  system += kcpustat_cpu(i).cpustat[CPUTIME_SYSTEM];
+  idle += get_idle_time(i);
+  iowait += get_iowait_time(i);
+  irq += kcpustat_cpu(i).cpustat[CPUTIME_IRQ];
```

```
+  softirq += kcpustat_cpu(i).cpustat[CPUTIME_SOFTIRQ];
+  steal += kcpustat_cpu(i).cpustat[CPUTIME_STEAL];
+  guest += kcpustat_cpu(i).cpustat[CPUTIME_GUEST];
+  guest_nice += kcpustat_cpu(i).cpustat[CPUTIME_GUEST_NICE];

   for (j = 0; j < NR_SOFTIRQS; j++) {
    unsigned int softirq_stat = kstat_softirqs_cpu(j, i);
@@ -106,16 +101,16 @@ static int show_stat(struct seq_file *p, void *v)
   (unsigned long long)cputime64_to_clock_t(guest_nice));
  for_each_online_cpu(i) {
   /* Copy values here to work around gcc-2.95.3, gcc-2.96 */
-  user = kstat_cpu(i).cpustat.user;
-  nice = kstat_cpu(i).cpustat.nice;
-  system = kstat_cpu(i).cpustat.system;
+  user = kcpustat_cpu(i).cpustat[CPUTIME_USER];
+  nice = kcpustat_cpu(i).cpustat[CPUTIME_NICE];
+  system = kcpustat_cpu(i).cpustat[CPUTIME_SYSTEM];
   idle = get_idle_time(i);
   iowait = get_iowait_time(i);
-  irq = kstat_cpu(i).cpustat.irq;
-  softirq = kstat_cpu(i).cpustat.softirq;
-  steal = kstat_cpu(i).cpustat.steal;
-  guest = kstat_cpu(i).cpustat.guest;
-  guest_nice = kstat_cpu(i).cpustat.guest_nice;
+  irq = kcpustat_cpu(i).cpustat[CPUTIME_IRQ];
+  softirq = kcpustat_cpu(i).cpustat[CPUTIME_SOFTIRQ];
+  steal = kcpustat_cpu(i).cpustat[CPUTIME_STEAL];
+  guest = kcpustat_cpu(i).cpustat[CPUTIME_GUEST];
+  guest_nice = kcpustat_cpu(i).cpustat[CPUTIME_GUEST_NICE];
   seq_printf(p,
    "cpu%d %llu %llu %llu %llu %llu %llu %llu %llu "
    "%llu\n",
diff --git a/fs/proc/uptime.c b/fs/proc/uptime.c
index 766b1d4..0fb22e4 100644
--- a/fs/proc/uptime.c
+++ b/fs/proc/uptime.c
@@ -12,10 +12,10 @@ static int uptime_proc_show(struct seq_file *m, void *v)
  struct timespec uptime;
  struct timespec idle;
  int i;
- cputime_t idletime = cputime_zero;
+ u64 idletime = 0;

  for_each_possible_cpu(i)
-  idletime = cputime64_add(idletime, kstat_cpu(i).cpustat.idle);
+  idletime += kcpustat_cpu(i).cpustat[CPUTIME_IDLE];

  do_posix_clock_monotonic_gettime(&uptime);
```

```
  monotonic_to_bootbased(&uptime);
diff --git a/include/linux/kernel_stat.h b/include/linux/kernel_stat.h
index 0cce2db..2fbd905 100644
--- a/include/linux/kernel_stat.h
+++ b/include/linux/kernel_stat.h
@@ -6,6 +6,7 @@
 #include <linux/percpu.h>
 #include <linux/cpumask.h>
 #include <linux/interrupt.h>
+#include <linux/sched.h>
 #include <asm/irq.h>
 #include <asm/cputime.h>

@@ -15,21 +16,25 @@
  * used by rstatd/perfmeter
  */

-struct cpu_usage_stat {
- cputime64_t user;
- cputime64_t nice;
- cputime64_t system;
- cputime64_t softirq;
- cputime64_t irq;
- cputime64_t idle;
- cputime64_t iowait;
- cputime64_t steal;
- cputime64_t guest;
- cputime64_t guest_nice;
+enum cpu_usage_stat {
+ CPUTIME_USER,
+ CPUTIME_NICE,
+ CPUTIME_SYSTEM,
+ CPUTIME_SOFTIRQ,
+ CPUTIME_IRQ,
+ CPUTIME_IDLE,
+ CPUTIME_IOWAIT,
+ CPUTIME_STEAL,
+ CPUTIME_GUEST,
+ CPUTIME_GUEST_NICE,
+ NR_STATS,
+};
+
+struct kernel_cpustat {
+ u64 cpustat[NR_STATS];
 };

 struct kernel_stat {
- struct cpu_usage_stat cpustat;
```

```
 #ifndef CONFIG_GENERIC_HARDIRQS
        unsigned int irqs[NR_IRQS];
 #endif
@@ -38,10 +43,13 @@ struct kernel_stat {
 };

 DECLARE_PER_CPU(struct kernel_stat, kstat);
+DECLARE_PER_CPU(struct kernel_cpustat, kernel_cpustat);

-#define kstat_cpu(cpu) per_cpu(kstat, cpu)
 /* Must have preemption disabled for this to be meaningful. */
-#define kstat_this_cpu __get_cpu_var(kstat)
+#define kstat_this_cpu (&__get_cpu_var(kstat))
+#define kcpustat_this_cpu (&__get_cpu_var(kernel_cpustat))
+#define kstat_cpu(cpu) per_cpu(kstat, cpu)
+#define kcpustat_cpu(cpu) per_cpu(kernel_cpustat, cpu)

 extern unsigned long long nr_context_switches(void);

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index ca8fd44..e9e0c19 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -895,14 +895,14 @@ static void update_rq_clock_task(struct rq *rq, s64 delta)
 #ifdef CONFIG_IRQ_TIME_ACCOUNTING
 static int irqtime_account_hi_update(void)
 {
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
  unsigned long flags;
  u64 latest_ns;
  int ret = 0;

  local_irq_save(flags);
  latest_ns = this_cpu_read(cpu_hardirq_time);
- if (cputime64_gt(nsecs_to_cputime64(latest_ns), cpustat->irq))
+ if (cputime64_gt(nsecs_to_cputime64(latest_ns), cpustat[CPUTIME_IRQ]))
   ret = 1;
  local_irq_restore(flags);
  return ret;
@@ -910,14 +910,14 @@ static int irqtime_account_hi_update(void)

 static int irqtime_account_si_update(void)
 {
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
  unsigned long flags;
  u64 latest_ns;
```

```
  int ret = 0;

  local_irq_save(flags);
  latest_ns = this_cpu_read(cpu_softirq_time);
- if (cputime64_gt(nsecs_to_cputime64(latest_ns), cpustat->softirq))
+ if (cputime64_gt(nsecs_to_cputime64(latest_ns), cpustat[CPUTIME_SOFTIRQ]))
   ret = 1;
  local_irq_restore(flags);
  return ret;
@@ -2499,8 +2499,10 @@ unlock:
 #endif

 DEFINE_PER_CPU(struct kernel_stat, kstat);
+DEFINE_PER_CPU(struct kernel_cpustat, kernel_cpustat);

 EXPORT_PER_CPU_SYMBOL(kstat);
+EXPORT_PER_CPU_SYMBOL(kernel_cpustat);

 /*
  * Return any ns on the sched_clock that have not yet been accounted in
@@ -2562,8 +2564,9 @@ unsigned long long task_sched_runtime(struct task_struct *p)
 void account_user_time(struct task_struct *p, cputime_t cputime,
        cputime_t cputime_scaled)
 {
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
- cputime64_t tmp;
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
+ u64 tmp;
+ int index;

  /* Add user time to process. */
  p->utime = cputime_add(p->utime, cputime);
@@ -2572,10 +2575,9 @@ void account_user_time(struct task_struct *p, cputime_t cputime,

  /* Add user time to cpustat. */
  tmp = cputime_to_cputime64(cputime);
- if (TASK_NICE(p) > 0)
-  cpustat->nice = cputime64_add(cpustat->nice, tmp);
- else
-  cpustat->user = cputime64_add(cpustat->user, tmp);
+
+ index = (TASK_NICE(p) > 0) ? CPUTIME_NICE : CPUTIME_USER;
+ cpustat[index] += tmp;

  cpuacct_update_stats(p, CPUACCT_STAT_USER, cputime);
  /* Account for user time used */
@@ -2591,8 +2593,8 @@ void account_user_time(struct task_struct *p, cputime_t cputime,
 static void account_guest_time(struct task_struct *p, cputime_t cputime,
```

```
          cputime_t cputime_scaled)
 {
- cputime64_t tmp;
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
+ u64 tmp;
+ u64 *cpustat = kcpustat_this_cpu->cpustat;

  tmp = cputime_to_cputime64(cputime);

@@ -2604,11 +2606,11 @@ static void account_guest_time(struct task_struct *p, cputime_t
cputime,

  /* Add guest time to cpustat. */
  if (TASK_NICE(p) > 0) {
-  cpustat->nice = cputime64_add(cpustat->nice, tmp);
-  cpustat->guest_nice = cputime64_add(cpustat->guest_nice, tmp);
+  cpustat[CPUTIME_NICE] += tmp;
+  cpustat[CPUTIME_GUEST_NICE] += tmp;
  } else {
-  cpustat->user = cputime64_add(cpustat->user, tmp);
-  cpustat->guest = cputime64_add(cpustat->guest, tmp);
+  cpustat[CPUTIME_USER] += tmp;
+  cpustat[CPUTIME_GUEST] += tmp;
  }
 }

@@ -2621,9 +2623,10 @@ static void account_guest_time(struct task_struct *p, cputime_t
cputime,
  */
 static inline
 void __account_system_time(struct task_struct *p, cputime_t cputime,
-   cputime_t cputime_scaled, cputime64_t *target_cputime64)
+   cputime_t cputime_scaled, int index)
 {
- cputime64_t tmp = cputime_to_cputime64(cputime);
+ u64 tmp = cputime_to_cputime64(cputime);
+ u64 *cpustat = kcpustat_this_cpu->cpustat;

  /* Add system time to process. */
  p->stime = cputime_add(p->stime, cputime);
@@ -2634,7 +2637,7 @@ void __account_system_time(struct task_struct *p, cputime_t cputime,
  account_group_system_time(p, cputime);

  /* Add system time to cpustat. */
- *target_cputime64 = cputime64_add(*target_cputime64, tmp);
+ cpustat[index] += tmp;
  cpuacct_update_stats(p, CPUACCT_STAT_SYSTEM, cputime);
```

```
  /* Account for system time used */
@@ -2648,8 +2651,7 @@ void __account_system_time(struct task_struct *p, cputime_t cputime,
 void account_system_time(struct task_struct *p, int hardirq_offset,
     cputime_t cputime, cputime_t cputime_scaled)
 {
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
- cputime64_t *target_cputime64;
+ int index;

  if ((p->flags & PF_VCPU) && (irq_count() - hardirq_offset == 0)) {
   account_guest_time(p, cputime, cputime_scaled);
@@ -2657,13 +2659,13 @@ void account_system_time(struct task_struct *p, int hardirq_offset,
  }

  if (hardirq_count() - hardirq_offset)
-  target_cputime64 = &cpustat->irq;
+  index = CPUTIME_IRQ;
  else if (in_serving_softirq())
-  target_cputime64 = &cpustat->softirq;
+  index = CPUTIME_SOFTIRQ;
  else
-  target_cputime64 = &cpustat->system;
+  index = CPUTIME_SYSTEM;

- __account_system_time(p, cputime, cputime_scaled, target_cputime64);
+ __account_system_time(p, cputime, cputime_scaled, index);
 }

 /*
@@ -2672,10 +2674,10 @@ void account_system_time(struct task_struct *p, int hardirq_offset,
  */
 void account_steal_time(cputime_t cputime)
 {
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
- cputime64_t cputime64 = cputime_to_cputime64(cputime);
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
+ u64 cputime64 = cputime_to_cputime64(cputime);

- cpustat->steal = cputime64_add(cpustat->steal, cputime64);
+ cpustat[CPUTIME_STEAL] += cputime64;
 }

 /*
@@ -2684,14 +2686,14 @@ void account_steal_time(cputime_t cputime)
  */
 void account_idle_time(cputime_t cputime)
 {
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
```

```
- cputime64_t cputime64 = cputime_to_cputime64(cputime);
+ u64 *cpustat = kcpustat_this_cpu->cpustat;
+ u64 cputime64 = cputime_to_cputime64(cputime);
  struct rq *rq = this_rq();

  if (atomic_read(&rq->nr_iowait) > 0)
-  cpustat->iowait = cputime64_add(cpustat->iowait, cputime64);
+  cpustat[CPUTIME_IOWAIT] += cputime64;
  else
-  cpustat->idle = cputime64_add(cpustat->idle, cputime64);
+  cpustat[CPUTIME_IDLE] += cputime64;
 }

 static __always_inline bool steal_account_process_tick(void)
@@ -2741,16 +2743,16 @@ static void irqtime_account_process_tick(struct task_struct *p, int
user_tick,
       struct rq *rq)
 {
  cputime_t one_jiffy_scaled = cputime_to_scaled(cputime_one_jiffy);
- cputime64_t tmp = cputime_to_cputime64(cputime_one_jiffy);
- struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
+ u64 tmp = cputime_to_cputime64(cputime_one_jiffy);
+ u64 *cpustat = kcpustat_this_cpu->cpustat;

  if (steal_account_process_tick())
   return;

  if (irqtime_account_hi_update()) {
-  cpustat->irq = cputime64_add(cpustat->irq, tmp);
+  cpustat[CPUTIME_IRQ] += tmp;
  } else if (irqtime_account_si_update()) {
-  cpustat->softirq = cputime64_add(cpustat->softirq, tmp);
+  cpustat[CPUTIME_SOFTIRQ] += tmp;
  } else if (this_cpu_ksoftirqd() == p) {
   /*
    * ksoftirqd time do not get accounted in cpu_softirq_time.
@@ -2758,7 +2760,7 @@ static void irqtime_account_process_tick(struct task_struct *p, int
user_tick,
    * Also, p->stime needs to be updated for ksoftirqd.
    */
   __account_system_time(p, cputime_one_jiffy, one_jiffy_scaled,
-    &cpustat->softirq);
+    CPUTIME_SOFTIRQ);
  } else if (user_tick) {
   account_user_time(p, cputime_one_jiffy, one_jiffy_scaled);
  } else if (p == rq->idle) {
@@ -2767,7 +2769,7 @@ static void irqtime_account_process_tick(struct task_struct *p, int
user_tick,
```

```
   account_guest_time(p, cputime_one_jiffy, one_jiffy_scaled);
  } else {
   __account_system_time(p, cputime_one_jiffy, one_jiffy_scaled,
-     &cpustat->system);
+     CPUTIME_SYSTEM);
  }
 }

--
1.7.6.4
```

Subject: [PATCH v2 2/3] Reuse cgroup's parent pointer
Posted by Glauber Costa on Mon, 28 Nov 2011 16:45:18 GMT
View Forum Message <> Reply to Message

We already have a pointer to the cgroup parent (whose data is more likely
to be in the cache than this, anyway), so there is no need to have this one
in cpuacct.

This patch makes the underlying cgroup be used instead.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Paul Tuner <pjt@google.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
---
 kernel/sched/core.c | 15 +++++++++------
 1 files changed, 9 insertions(+), 6 deletions(-)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index e9e0c19..c36f926 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -7840,7 +7840,6 @@ struct cpuacct {
  /* cpuusage holds pointer to a u64-type object on every cpu */
  u64 __percpu *cpuusage;
  struct percpu_counter cpustat[CPUACCT_STAT_NSTATS];
- struct cpuacct *parent;
 };

 struct cgroup_subsys cpuacct_subsys;
@@ -7859,6 +7858,13 @@ static inline struct cpuacct *task_ca(struct task_struct *tsk)
        struct cpuacct, css);
 }

+static inline struct cpuacct *parent_ca(struct cpuacct *ca)
+{
```

```
+ if (!ca || !ca->css.cgroup->parent)
+  return NULL;
+ return cgroup_ca(ca->css.cgroup->parent);
+}
+
 /* create a new cpu accounting group */
 static struct cgroup_subsys_state *cpuacct_create(
  struct cgroup_subsys *ss, struct cgroup *cgrp)
@@ -7877,9 +7883,6 @@ static struct cgroup_subsys_state *cpuacct_create(
   if (percpu_counter_init(&ca->cpustat[i], 0))
    goto out_free_counters;

- if (cgrp->parent)
-  ca->parent = cgroup_ca(cgrp->parent);
-
  return &ca->css;

 out_free_counters:
@@ -8046,7 +8049,7 @@ void cpuacct_charge(struct task_struct *tsk, u64 cputime)

  ca = task_ca(tsk);

- for (; ca; ca = ca->parent) {
+ for (; ca; ca = parent_ca(ca)) {
   u64 *cpuusage = per_cpu_ptr(ca->cpuusage, cpu);
   *cpuusage += cputime;
  }
@@ -8088,7 +8091,7 @@ void cpuacct_update_stats(struct task_struct *tsk,

  do {
   __percpu_counter_add(&ca->cpustat[idx], val, batch);
-  ca = ca->parent;
+  ca = parent_ca(ca);
  } while (ca);
  rcu_read_unlock();
 }
--
1.7.6.4
```

Subject: [PATCH v2 3/3] cpuacct.stat: re-use scheduler statistics for the root cgroup

Posted by Glauber Costa on Mon, 28 Nov 2011 16:45:19 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/core.c | 165 ++++++++++++++++++++++++++++++++----------------
 kernel/sched/sched.h | 33 ++++++++-
 2 files changed, 105 insertions(+), 93 deletions(-)

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index c36f926..fd73bea 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -2555,6 +2555,42 @@ unsigned long long task_sched_runtime(struct task_struct *p)
  return ns;
 }

+#ifdef CONFIG_CGROUP_CPUACCT
+struct cgroup_subsys cpuacct_subsys;
+struct cpuacct root_cpuacct;
+#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
+}
+
+
 /*
  * Account user cpu time to a process.
  * @p: the process that the cpu time gets accounted to
@@ -2579,7 +2615,7 @@ 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);
 }
@@ -2635,7 +2671,7 @@ 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);
@@ -6772,8 +6808,15 @@ void __init sched_init(void)
  INIT_LIST_HEAD(&root_task_group.children);
  INIT_LIST_HEAD(&root_task_group.siblings);
  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;
```

```
@@ -7834,44 +7877,16 @@ struct cgroup_subsys cpu_cgroup_subsys = {
  * (balbir@in.ibm.com).
  */

-/* track cpu usage of a group of tasks and its child groups */
-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 cgroup_subsys cpuacct_subsys;
-
-/* return cpu accounting group corresponding to this container */
-static inline struct cpuacct *cgroup_ca(struct cgroup *cgrp)
-{
- return container_of(cgroup_subsys_state(cgrp, 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);
-}
-
-static inline struct cpuacct *parent_ca(struct cpuacct *ca)
-{
- if (!ca || !ca->css.cgroup->parent)
-  return NULL;
- return cgroup_ca(ca->css.cgroup->parent);
-}
-
 /* create a new cpu accounting group */
 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;

@@ -7879,15 +7894,13 @@ 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);
@@ -7900,10 +7913,8 @@ 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);
 }
@@ -7996,16 +8007,31 @@ 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 (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);
+ 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);
+
+ 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;
 }

@@ -8057,45 +8083,6 @@ void cpuacct_charge(struct task_struct *tsk, u64 cputime)
  rcu_read_unlock();
 }

-/*
- * When CONFIG_VIRT_CPU_ACCOUNTING is enabled one jiffy can be very large
- * in cputime_t units. As a result, cpuacct_update_stats calls
- * percpu_counter_add with values large enough to always overflow the
- * per cpu batch limit causing bad SMP scalability.
- *
- * To fix this we scale percpu_counter_batch by cputime_one_jiffy so we
- * batch the same amount of time with CONFIG_VIRT_CPU_ACCOUNTING disabled
- * and enabled. We cap it at INT_MAX which is the largest allowed batch value.
- */
-#ifdef CONFIG_SMP
-#define CPUACCT_BATCH \
- min_t(long, percpu_counter_batch * cputime_one_jiffy, INT_MAX)
-#else
-#define CPUACCT_BATCH 0
-#endif
-
-/*
- * Charge the system/user time to the task's accounting group.
- */
```

```
-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,
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index c2e7802..53f6f83 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -828,13 +828,38 @@ extern void init_rt_bandwidth(struct rt_bandwidth *rt_b, u64 period,
u64 runtime
 extern void update_cpu_load(struct rq *this_rq);

 #ifdef CONFIG_CGROUP_CPUACCT
+/* track cpu usage of a group of tasks and its child groups */
+struct cpuacct {
+ struct cgroup_subsys_state css;
+ /* cpuusage holds pointer to a u64-type object on every cpu */
+ u64 __percpu *cpuusage;
+ struct kernel_cpustat __percpu *cpustat;
+};
+
+/* return cpu accounting group corresponding to this container */
+static inline struct cpuacct *cgroup_ca(struct cgroup *cgrp)
+{
+ return container_of(cgroup_subsys_state(cgrp, 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);
+}
+
+static inline struct cpuacct *parent_ca(struct cpuacct *ca)
+{
+ if (!ca || !ca->css.cgroup->parent)
+  return NULL;
+ return cgroup_ca(ca->css.cgroup->parent);
+}
+
 extern void cpuacct_charge(struct task_struct *tsk, u64 cputime);
-extern 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 inc_nr_running(struct rq *rq)
--
1.7.6.4
```

Subject: Re: [PATCH v2 0/3] cpuacct cgroup refactoring
Posted by Peter Zijlstra on Thu, 01 Dec 2011 10:56:59 GMT
View Forum Message <> Reply to Message

On Mon, 2011-11-28 at 14:45 -0200, Glauber Costa wrote:
> Hi,
>
> This is the same series I've already sent before. Just sending again
> after the conflict with the code that was in tip.git

Utter lack of complaints here.. lets just merge it :-)

Should show up in:

 git://git.kernel.org/pub/scm/linux/kernel/git/peterz/queue.g it

soonish.
