Subject: [PATCH v4 0/4] per cgroup cpu statistics
Posted by Glauber Costa on Tue, 05 Jun 2012 14:49:37 GMT
View Forum Message <> Reply to Message

Hello,

This is the 4th version of this per-cgroup statistics. My aim with that is
to provide userspace with the necessary tools to export a view of /proc/stat
inside a container, so tools like top would work.

I believe this version merges all the comments from Peter and Paul. Let me know
if I forgot any of them.

I now reuse put_prev_task to account for nr_switches, so no extra walks are added.
For that, I had to add another parameter to the function, but I hope this is
acceptable. Please note that this parameter would still be needed even if
Peter's patch that merges put_prev_task behavior inside pick_next_task.

This is because the previous class may live in a different class and we may
have to call it anyway. In which case a hint like this would be needed.
I believe this to be orthogonal to the work you are doing.

Peter: the patch that adds exec_clock to rt was removed. That was my bad, it
was intended to be part of the series that unifies cpu and cpuacct for the
comounted case. I have them together in my tree, and I made the cut in the wrong
place. It is not needed here.

Let me know what you think of this.

v4:
* read_seq_string used instead of read_map. This is because the
  buffer can get very big, and that's easier to fix by using seq_string
* idle time no longer exported. It can be derived from userspace easily
* "steal" changed to "wait", since steal is more our interpretation of it
* nr_switches now being accounted as we walk the tree in put_prev_task, so no
  new hierarchy walks are being inserted.
v3:
* completely reworked nr_switches gathering
* separated per-se sleep_start to be more clear about it

Glauber Costa (4):
  account guest time per-cgroup as well.
  expose fine-grained per-cpu data for cpuacct stats
  mark whenever put_prev_task is called from context_switch
  expose per-taskgroup schedstats in cgroup

 include/linux/sched.h | 2 +-

```
 kernel/sched/core.c      | 178 +++++++++++++++++++++++++++++++++++++++++++++---
 kernel/sched/fair.c      |  28 +++++++-
 kernel/sched/idle_task.c |   2 +-
 kernel/sched/rt.c        |   6 +-
 kernel/sched/sched.h     |   5 ++
 kernel/sched/stop_task.c |   2 +-
 7 files changed, 207 insertions(+), 16 deletions(-)

--
1.7.10.2
```

Subject: [PATCH v4 1/4] account guest time per-cgroup as well.
Posted by Glauber Costa on Tue, 05 Jun 2012 14:49:38 GMT
View Forum Message <> Reply to Message

We already track multiple tick statistics per-cgroup, using
the task_group_account_field facility. This patch accounts
guest_time in that manner as well.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/core.c | 10 ++++++------
 1 file changed, 4 insertions(+), 6 deletions(-)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 39eb601..220d416 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -2717,8 +2717,6 @@ 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)
 {
- u64 *cpustat = kcpustat_this_cpu->cpustat;
-
   /* Add guest time to process. */
   p->utime += cputime;
   p->utimescaled += cputime_scaled;
@@ -2727,11 +2725,11 @@ static void account_guest_time(struct task_struct *p, cputime_t
cputime,

   /* Add guest time to cpustat. */
   if (TASK_NICE(p) > 0) {
-  cpustat[CPUTIME_NICE] += (__force u64) cputime;
-  cpustat[CPUTIME_GUEST_NICE] += (__force u64) cputime;
+  task_group_account_field(p, CPUTIME_NICE, (__force u64) cputime);
```

```
+  task_group_account_field(p, CPUTIME_GUEST, (__force u64) cputime);
  } else {
-  cpustat[CPUTIME_USER] += (__force u64) cputime;
-  cpustat[CPUTIME_GUEST] += (__force u64) cputime;
+  task_group_account_field(p, CPUTIME_USER, (__force u64) cputime);
+  task_group_account_field(p, CPUTIME_GUEST, (__force u64) cputime);
  }
 }

--
1.7.10.2
```

Subject: [PATCH v4 2/4] expose fine-grained per-cpu data for cpuacct stats
Posted by Glauber Costa on Tue, 05 Jun 2012 14:49:39 GMT
View Forum Message <> Reply to Message

The cpuacct cgroup already exposes user and system numbers in a per-cgroup fashion. But they are a summation along the whole group, not a per-cpu figure. Also, they are coarse-grained version of the stats usually shown at places like /proc/stat.

I want to have enough cgroup data to emulate the /proc/stat interface. To achieve that, I am creating a new file "stat_percpu" that displays the fine-grained per-cpu data. The original data is left alone.

The format of this file is as follows:

1line header
cpux val1 val2 ... valn.

It is the same format used for the cpu part /proc/stat, except for the header, that may allow us to add fields in the future if they prove themselves needed.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/core.c | 37 ++
 1 file changed, 37 insertions(+)

```
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 220d416..c058189 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -8178,6 +8178,39 @@ static int cpuacct_stats_show(struct cgroup *cgrp, struct cftype *cft,
  return 0;
 }
```

```
+static inline void do_fill_seq(struct seq_file *m, struct cpuacct *ca,
+          int cpu, int index)
+{
+ struct kernel_cpustat *kcpustat = per_cpu_ptr(ca->cpustat, cpu);
+ u64 val;
+
+ val = cputime64_to_clock_t(kcpustat->cpustat[index]);
+ seq_put_decimal_ull(m, ' ', val);
+}
+
+static int cpuacct_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
+         struct seq_file *m)
+{
+ struct cpuacct *ca = cgroup_ca(cgrp);
+ int cpu;
+
+ seq_printf(m, "user nice system irq softirq guest guest_nice\n");
+
+ for_each_online_cpu(cpu) {
+  seq_printf(m, "cpu%d", cpu);
+  do_fill_seq(m, ca, cpu, CPUTIME_USER);
+  do_fill_seq(m, ca, cpu, CPUTIME_NICE);
+  do_fill_seq(m, ca, cpu, CPUTIME_SYSTEM);
+  do_fill_seq(m, ca, cpu, CPUTIME_IRQ);
+  do_fill_seq(m, ca, cpu, CPUTIME_SOFTIRQ);
+  do_fill_seq(m, ca, cpu, CPUTIME_GUEST);
+  do_fill_seq(m, ca, cpu, CPUTIME_GUEST_NICE);
+  seq_putc(m, '\n');
+ }
+
+ return 0;
+}
+
 static struct cftype files[] = {
  {
   .name = "usage",
@@ -8192,6 +8225,10 @@ static struct cftype files[] = {
   .name = "stat",
   .read_map = cpuacct_stats_show,
  },
+ {
+  .name = "stat_percpu",
+  .read_seq_string = cpuacct_stats_percpu_show,
+ },
  { } /* terminate */
 };
```

--
1.7.10.2

Subject: [PATCH v4 3/4] mark whenever put_prev_task is called from context_switch
Posted by Glauber Costa on Tue, 05 Jun 2012 14:49:40 GMT
View Forum Message <> Reply to Message

In the interest of having the number of context switches stored per-group,
this patch introduces a boolean argument to put_prev_task. It is needed
because put_prev_task is called from many sites, not all of them denoting
a context switch.

This would be needed even if pick_next_task has the behavior of put_prev_task
embedded into it: This is because we may be picking a task from a different
class, and that would translate to a raw call to put_prev_task.

For the fair class, nr_switches is stored for all the hierarchy. This is
because a walk is done anyway there, so we just reuse it.

For the rt class, we only store in the current group, therefore not
forcing a walk. Readers then may have to accumulate.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 include/linux/sched.h | 2 +-
 kernel/sched/core.c | 11 ++++++++-----
 kernel/sched/fair.c | 4 +++-
 kernel/sched/idle_task.c | 2 +-
 kernel/sched/rt.c | 6 +++++-
 kernel/sched/sched.h | 3 +++
 kernel/sched/stop_task.c | 2 +-
 7 files changed, 20 insertions(+), 10 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index f45c0b2..2bff097 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1083,7 +1083,7 @@ struct sched_class {
  void (*check_preempt_curr) (struct rq *rq, struct task_struct *p, int flags);

  struct task_struct * (*pick_next_task) (struct rq *rq);
- void (*put_prev_task) (struct rq *rq, struct task_struct *p);
+ void (*put_prev_task) (struct rq *rq, struct task_struct *p, bool ctxsw);

```
 #ifdef CONFIG_SMP
  int  (*select_task_rq)(struct task_struct *p, int sd_flag, int flags);
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index c058189..6803fd1 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -3164,7 +3164,7 @@ static void put_prev_task(struct rq *rq, struct task_struct *prev)
 {
  if (prev->on_rq || rq->skip_clock_update < 0)
   update_rq_clock(rq);
- prev->sched_class->put_prev_task(rq, prev);
+ prev->sched_class->put_prev_task(rq, prev, true);
 }

 /*
@@ -3248,6 +3248,7 @@ need_resched:
  if (unlikely(!rq->nr_running))
   idle_balance(cpu, rq);

+
  put_prev_task(rq, prev);
  next = pick_next_task(rq);
  clear_tsk_need_resched(prev);
@@ -3857,7 +3858,7 @@ void rt_mutex_setprio(struct task_struct *p, int prio)
  if (on_rq)
   dequeue_task(rq, p, 0);
  if (running)
-  p->sched_class->put_prev_task(rq, p);
+  p->sched_class->put_prev_task(rq, p, false);

  if (rt_prio(prio))
   p->sched_class = &rt_sched_class;
@@ -4208,7 +4209,7 @@ recheck:
  if (on_rq)
   dequeue_task(rq, p, 0);
  if (running)
-  p->sched_class->put_prev_task(rq, p);
+  p->sched_class->put_prev_task(rq, p, false);

  p->sched_reset_on_fork = reset_on_fork;

@@ -5197,7 +5198,7 @@ static void migrate_tasks(unsigned int dead_cpu)

   next = pick_next_task(rq);
   BUG_ON(!next);
-  next->sched_class->put_prev_task(rq, next);
+  next->sched_class->put_prev_task(rq, next, false);
```

```
   /* Find suitable destination for @next, with force if needed. */
   dest_cpu = select_fallback_rq(dead_cpu, next);
@@ -7318,7 +7319,7 @@ void sched_move_task(struct task_struct *tsk)
  if (on_rq)
   dequeue_task(rq, tsk, 0);
  if (unlikely(running))
-  tsk->sched_class->put_prev_task(rq, tsk);
+  tsk->sched_class->put_prev_task(rq, tsk, false);

 #ifdef CONFIG_FAIR_GROUP_SCHED
  if (tsk->sched_class->task_move_group)
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 940e6d1..1c8a04e 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -2996,7 +2996,7 @@ static struct task_struct *pick_next_task_fair(struct rq *rq)
 /*
  * Account for a descheduled task:
  */
-static void put_prev_task_fair(struct rq *rq, struct task_struct *prev)
+static void put_prev_task_fair(struct rq *rq, struct task_struct *prev, bool ctxsw)
 {
  struct sched_entity *se = &prev->se;
  struct cfs_rq *cfs_rq;
@@ -3004,6 +3004,8 @@ static void put_prev_task_fair(struct rq *rq, struct task_struct *prev)
  for_each_sched_entity(se) {
   cfs_rq = cfs_rq_of(se);
   put_prev_entity(cfs_rq, se);
+  if (ctxsw)
+   cfs_rq->nr_switches++;
  }
 }

diff --git a/kernel/sched/idle_task.c b/kernel/sched/idle_task.c
index b44d604..08c3ad3 100644
--- a/kernel/sched/idle_task.c
+++ b/kernel/sched/idle_task.c
@@ -42,7 +42,7 @@ dequeue_task_idle(struct rq *rq, struct task_struct *p, int flags)
  raw_spin_lock_irq(&rq->lock);
 }

-static void put_prev_task_idle(struct rq *rq, struct task_struct *prev)
+static void put_prev_task_idle(struct rq *rq, struct task_struct *prev, bool ctxsw)
 {
 }

diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
index c5565c3..afe5096 100644
```

```
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -1387,8 +1387,10 @@ static struct task_struct *pick_next_task_rt(struct rq *rq)
  return p;
 }

-static void put_prev_task_rt(struct rq *rq, struct task_struct *p)
+static void put_prev_task_rt(struct rq *rq, struct task_struct *p, bool ctxsw)
 {
+ struct sched_rt_entity *rt_se = &p->rt;
+ struct rt_rq *rt_rq = rt_rq_of_se(rt_se);
  update_curr_rt(rq);

  /*
@@ -1397,6 +1399,8 @@ static void put_prev_task_rt(struct rq *rq, struct task_struct *p)
   */
  if (on_rt_rq(&p->rt) && p->rt.nr_cpus_allowed > 1)
   enqueue_pushable_task(rq, p);
+ if (ctxsw)
+  rt_rq->rt_nr_switches++;
 }

 #ifdef CONFIG_SMP
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index ba9dccf..7548814 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -237,6 +237,7 @@ struct cfs_rq {
  struct list_head leaf_cfs_rq_list;
  struct task_group *tg; /* group that "owns" this runqueue */

+ u64 nr_switches;
 #ifdef CONFIG_SMP
  /*
   *   h_load = weight * f(tg)
@@ -306,6 +307,8 @@ struct rt_rq {
  struct rq *rq;
  struct list_head leaf_rt_rq_list;
  struct task_group *tg;
+
+ u64 rt_nr_switches;
 #endif
 };

diff --git a/kernel/sched/stop_task.c b/kernel/sched/stop_task.c
index 7b386e8..19ff22d 100644
--- a/kernel/sched/stop_task.c
+++ b/kernel/sched/stop_task.c
```

```
@@ -50,7 +50,7 @@ static void yield_task_stop(struct rq *rq)
  BUG(); /* the stop task should never yield, its pointless. */
 }

-static void put_prev_task_stop(struct rq *rq, struct task_struct *prev)
+static void put_prev_task_stop(struct rq *rq, struct task_struct *prev, bool ctxsw)
 {
 }

--
1.7.10.2
```

Subject: [PATCH v4 4/4] expose per-taskgroup schedstats in cgroup
Posted by Glauber Costa on Tue, 05 Jun 2012 14:49:41 GMT
View Forum Message <> Reply to Message

This patch aims at exposing stat information per-cgroup, such as:
 * steal time (rq wait time),
 * # context switches
 * # process running

The ultimate goal is to be able to present a per-container view of
/proc/stat inside a container. With this patch, everything that is needed to do
that is in place, except for number of tasks.

For most of the data, I achieve that by hooking into the schedstats framework,
so although the overhead of that is prone to discussion, I am not adding anything,
but reusing what's already there instead. The exception being that the data is
now computed and stored in non-task se's as well, instead of entity_is_task() branches.
However, I expect this to be minimum comparing to the alternative of adding new
hierarchy walks. Those are kept intact.

The format of the new file added is the same as the one recently
introduced for cpuacct:

1line header
cpux val1 val2 ... valn.

It is the same format used for the cpu part /proc/stat, except for the header,
that may allow us to add fields in the future if they prove themselves needed.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
 kernel/sched/core.c | 120 ++
 kernel/sched/fair.c | 24 ++++++++++

```
 kernel/sched/sched.h |    2 +
 3 files changed, 146 insertions(+)

diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 6803fd1..59b4466 100644
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -7961,6 +7961,113 @@ static u64 cpu_rt_period_read_uint(struct cgroup *cgrp, struct cftype
*cft)
 }
 #endif /* CONFIG_RT_GROUP_SCHED */

+#ifdef CONFIG_SCHEDSTATS
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+#define fair_rq(field, tg, i)  tg->cfs_rq[i]->field
+#else
+#define fair_rq(field, tg, i)  0
+#endif
+
+#ifdef CONFIG_RT_GROUP_SCHED
+#define rt_rq(field, tg, i)  tg->rt_rq[i]->field
+
+struct rt_switches_data {
+ u64 *switches;
+ int cpu;
+};
+
+static int tg_rt_count_switches(struct task_group *tg, void *data)
+{
+ struct rt_switches_data *switches = data;
+
+ *(switches->switches) += rt_rq(rt_nr_switches, tg, switches->cpu);
+ return 0;
+}
+
+static u64 tg_nr_rt_switches(struct task_group *tg, int cpu)
+{
+ u64 switches = 0;
+
+ struct rt_switches_data data = {
+  .switches = &switches,
+  .cpu = cpu,
+ };
+
+ rcu_read_lock();
+ walk_tg_tree_from(tg, tg_rt_count_switches, tg_nop, &data);
+ rcu_read_unlock();
```

```
+ return switches;
+}
+#else
+static u64 tg_nr_rt_switches(struct task_group *tg, int cpu)
+{
+ return 0;
+}
+#define rt_rq(field, tg, i)  0
+#endif
+
+static u64 tg_nr_switches(struct task_group *tg, int cpu)
+{
+
+ if (tg == &root_task_group)
+  return cpu_rq(cpu)->nr_switches;
+
+ return fair_rq(nr_switches, tg, cpu) + tg_nr_rt_switches(tg, cpu);
+}
+
+static u64 tg_nr_running(struct task_group *tg, int cpu)
+{
+ /*
+  * because of autogrouped groups in root_task_group, the
+  * following does not hold.
+  */
+ if (tg != &root_task_group)
+  return rt_rq(rt_nr_running, tg, cpu) + fair_rq(nr_running, tg, cpu);
+
+ return cpu_rq(cpu)->nr_running;
+}
+
+static u64 tg_wait(struct task_group *tg, int cpu)
+{
+ u64 val;
+
+ if (tg != &root_task_group)
+  val = cfs_read_wait(tg->se[cpu]);
+ else
+  /*
+   * There are many errors here that we are accumulating.
+   * However, we only provide this in the interest of having
+   * a consistent interface for all cgroups. Everybody
+   * probing the root cgroup should be getting its figures
+   * from system-wide files as /proc/stat. That would be faster
+   * to begin with...
+   */
+  val = kcpustat_cpu(cpu).cpustat[CPUTIME_STEAL] * TICK_NSEC;
+
```

```
+ return val;
+}
+
+static int cpu_stats_percpu_show(struct cgroup *cgrp, struct cftype *cft,
+     struct seq_file *m)
+{
+ struct task_group *tg = cgroup_tg(cgrp);
+ int cpu;
+
+ seq_printf(m, "wait nr_switches nr_running\n");
+
+ for_each_online_cpu(cpu) {
+  seq_printf(m, "cpu%d", cpu);
+  seq_put_decimal_ull(m, ' ', tg_wait(tg, cpu));
+  seq_put_decimal_ull(m, ' ', tg_nr_switches(tg, cpu));
+  seq_put_decimal_ull(m, ' ', tg_nr_running(tg, cpu));
+  seq_putc(m, '\n');
+ }
+
+ return 0;
+}
+#endif
+
 static struct cftype cpu_files[] = {
 #ifdef CONFIG_FAIR_GROUP_SCHED
  {
@@ -7968,6 +8075,19 @@ static struct cftype cpu_files[] = {
   .read_u64 = cpu_shares_read_u64,
   .write_u64 = cpu_shares_write_u64,
  },
+/*
+ * In theory, those could be done using the rt tasks as a basis
+ * as well. Since we're interested in figures like idle, iowait, etc
+ * for the whole cgroup, the results should be the same.
+ * But that only complicates the code, and I doubt anyone using !FAIR_GROUP_SCHED
+ * is terribly interested in those.
+ */
+#ifdef CONFIG_SCHEDSTATS
+ {
+  .name = "stat_percpu",
+  .read_seq_string = cpu_stats_percpu_show,
+ },
+#endif
 #endif
 #ifdef CONFIG_CFS_BANDWIDTH
  {
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 1c8a04e..c842d6a 100644
```

```
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -719,6 +719,30 @@ update_stats_wait_start(struct cfs_rq *cfs_rq, struct sched_entity *se)
  schedstat_set(se->statistics.wait_start, rq_of(cfs_rq)->clock);
 }

+#ifdef CONFIG_SCHEDSTATS
+u64 cfs_read_sleep(struct sched_entity *se)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.sum_sleep_runtime;
+
+ if (!se->statistics.sleep_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.sleep_start;
+}
+
+u64 cfs_read_wait(struct sched_entity *se)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ u64 value = se->statistics.wait_sum;
+
+ if (!se->statistics.wait_start)
+  return value;
+
+ return value + rq_of(cfs_rq)->clock - se->statistics.wait_start;
+}
+#endif
+
 /*
  * Task is being enqueued - update stats:
  */
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index 7548814..768680d 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -1149,6 +1149,8 @@ extern void init_rt_rq(struct rt_rq *rt_rq, struct rq *rq);
 extern void unthrottle_offline_cfs_rqs(struct rq *rq);

 extern void account_cfs_bandwidth_used(int enabled, int was_enabled);
+extern u64 cfs_read_sleep(struct sched_entity *se);
+extern u64 cfs_read_wait(struct sched_entity *se);

 #ifdef CONFIG_NO_HZ
 enum rq_nohz_flag_bits {
--
1.7.10.2
```

Subject: Re: [PATCH v4 0/4] per cgroup cpu statistics
Posted by Glauber Costa on Thu, 14 Jun 2012 12:18:45 GMT
View Forum Message <> Reply to Message

On 06/05/2012 06:49 PM, Glauber Costa wrote:
> Hello,
>
> This is the 4th version of this per-cgroup statistics. My aim with that is
> to provide userspace with the necessary tools to export a view of /proc/stat
> inside a container, so tools like top would work.
>
> I believe this version merges all the comments from Peter and Paul. Let me know
> if I forgot any of them.
>
> I now reuse put_prev_task to account for nr_switches, so no extra walks are added.
> For that, I had to add another parameter to the function, but I hope this is
> acceptable. Please note that this parameter would still be needed even if
> Peter's patch that merges put_prev_task behavior inside pick_next_task.
>
> This is because the previous class may live in a different class and we may
> have to call it anyway. In which case a hint like this would be needed.
> I believe this to be orthogonal to the work you are doing.
>
> Peter: the patch that adds exec_clock to rt was removed. That was my bad, it
> was intended to be part of the series that unifies cpu and cpuacct for the
> comounted case. I have them together in my tree, and I made the cut in the wrong
> place. It is not needed here.
>
> Let me know what you think of this.
>
> v4:
> * read_seq_string used instead of read_map. This is because the
> buffer can get very big, and that's easier to fix by using seq_string
> * idle time no longer exported. It can be derived from userspace easily
> * "steal" changed to "wait", since steal is more our interpretation of it
> * nr_switches now being accounted as we walk the tree in put_prev_task, so no
> new hierarchy walks are being inserted.
> v3:
> * completely reworked nr_switches gathering
> * separated per-se sleep_start to be more clear about it
>
>
Any comments in this incarnation of the series?