Subject: [Patch 2/3] cpu controller (v3) - based on RTLIMIT_RT_CPU patch
Posted by Srivatsa Vaddagiri on Fri, 01 Dec 2006 16:51:25 GMT
View Forum Message <> Reply to Message

Nick/Ingo,

Here's another approach for a minimal cpu controller. Would greatly appreciate any feedback as before.

This version is inspired by Ingo's RTLIMIT_RT_CPU patches found here:

http://kernel.org/pub/linux/kernel/people/akpm/patches/2.6/2.6.11-rc2/2.6.11-rc2-mm2/broken-out/rlimit_rt_cpu.patch

This patch is also about 80% smaller than the patches I had posted earlier:

http://lkml.org/lkml/2006/9/28/236

Primary differences between Ingo's RT_LIMIT_CPU patch and this one:

- This patch handles starvation of lower priority tasks in a group.
- This patch uses tokens for accounting cpu consumption. I didnt get good results with decaying avg approach used in the rtlimit patches.
- Task grouping based on cpuset/containers and not on rtlimit.

Other features:

- Retains one-runqueue-per-cpu, as is prevalent today
- scheduling no longer of O(1) complexity, similar to rtlimit patches. This can be avoided if we use separate runqueues for different groups (as was done in http://lkml.org/lkml/2006/9/28/236)
- Only limit supported (no guarantee)

Unsupported feature:

- SMP load balance aware of group limits. This can be handled using smpnice later if required (http://lkml.org/lkml/2006/9/28/244)

Signed-off-by : Srivatsa Vaddagiri <vatsa@in.ibm.com>

---

linux-2.6.19-rc6-vatsa/include/linux/sched.h | 3
linux-2.6.19-rc6-vatsa/kernel/sched.c | 195 ++++++++++++++++++++++++++++++++++++-
2 files changed, 195 insertions(+), 3 deletions(-)

diff -puN include/linux/sched.h~cpu_ctlr include/linux/sched.h

```
--- linux-2.6.19-rc6/include/linux/sched.h~cpu_ctlr 2006-12-01 20:45:08.000000000 +0530
+++ linux-2.6.19-rc6-vatsa/include/linux/sched.h 2006-12-01 20:45:08.000000000 +0530
@@ -1095,6 +1095,9 @@ static inline void put_task_struct(struc
      /* Not implemented yet, only for 486*/
 #define PF_STARTING 0x00000002 /* being created */
 #define PF_EXITING 0x00000004 /* getting shut down */
+#ifdef CONFIG_CPUMETER
+#define PF_STARVING 0x00000010      /* Task starving for CPU */
+#endif
 #define PF_FORKNOEXEC 0x00000040 /* forked but didn't exec */
 #define PF_SUPERPRIV 0x00000100 /* used super-user privileges */
 #define PF_DUMPCORE 0x00000200 /* dumped core */
diff -puN kernel/sched.c~cpu_ctlr kernel/sched.c
--- linux-2.6.19-rc6/kernel/sched.c~cpu_ctlr 2006-12-01 20:45:08.000000000 +0530
+++ linux-2.6.19-rc6-vatsa/kernel/sched.c 2006-12-01 20:45:08.000000000 +0530
@@ -264,10 +264,25 @@ struct rq {
  unsigned long ttwu_local;
 #endif
  struct lock_class_key rq_lock_key;
+#ifdef CONFIG_CPUMETER
+ unsigned long last_update;
+ int need_recheck;
+#endif
 };

 static DEFINE_PER_CPU(struct rq, runqueues);

+struct cpu_usage {
+ long tokens;
+ unsigned long last_update;
+ int starve_count;
+};
+
+struct task_grp {
+ unsigned long limit;
+ struct cpu_usage *cpu_usage; /* per-cpu ptr */
+};
+
 static inline int cpu_of(struct rq *rq)
 {
 #ifdef CONFIG_SMP
@@ -705,6 +720,137 @@ enqueue_task_head(struct task_struct *p,
  p->array = array;
 }

+#ifdef CONFIG_CPUMETER
+
+#define task_starving(p) (p->flags & PF_STARVING)
```

```
+
+static inline struct task_grp *task_grp(struct task_struct *p)
+{
+ return NULL;
+}
+
+/* Mark a task starving - either we shortcircuited its timeslice or we didnt
+ * pick it to run (because its group ran out of bandwidth limit).
+ */
+static inline void set_tsk_starving(struct task_struct *p, struct task_grp *grp)
+{
+ struct cpu_usage *grp_usage;
+
+ if (task_starving(p))
+  return;
+
+ BUG_ON(!grp);
+ grp_usage = per_cpu_ptr(grp->cpu_usage, task_cpu(p));
+ grp_usage->starve_count++;
+ p->flags |= PF_STARVING;
+}
+
+/* Clear a task's starving flag */
+static inline void clear_tsk_starving(struct task_struct *p,
+       struct task_grp *grp)
+{
+ struct cpu_usage *grp_usage;
+
+ if (!task_starving(p))
+  return;
+
+ BUG_ON(!grp);
+ grp_usage = per_cpu_ptr(grp->cpu_usage, task_cpu(p));
+ grp_usage->starve_count--;
+ p->flags &= ~PF_STARVING;
+}
+
+/* Does the task's group have starving tasks? */
+static inline int is_grp_starving(struct task_struct *p)
+{
+ struct task_grp *grp = task_grp(p);
+ struct cpu_usage *grp_usage;
+
+ if (!grp)
+  return 0;
+
+ grp_usage = per_cpu_ptr(grp->cpu_usage, task_cpu(p));
+ if (grp_usage->starve_count)
```

```
+  return 1;
+
+ return 0;
+}
+
+/* Are we past the 1-sec control window? If so, all groups get to renew their
+ * expired tokens.
+ */
+static inline void adjust_control_window(struct task_struct *p)
+{
+ struct rq *rq = task_rq(p);
+ unsigned long delta;
+
+ delta = jiffies - rq->last_update;
+ if (delta >= HZ) {
+  rq->last_update += (delta/HZ) * HZ;
+  rq->need_recheck = 1;
+ }
+}
+
+/* Account group's cpu usage */
+static inline void inc_cpu_usage(struct task_struct *p)
+{
+ struct task_grp *grp = task_grp(p);
+ struct cpu_usage *grp_usage;
+
+ adjust_control_window(p);
+
+ if (!grp || !grp->limit || rt_task(p))
+  return;
+
+ grp_usage = per_cpu_ptr(grp->cpu_usage, task_cpu(p));
+ grp_usage->tokens--;
+}
+
+static inline int task_over_cpu_limit(struct task_struct *p)
+{
+ struct rq *rq = task_rq(p);
+ struct task_grp *grp = task_grp(p);
+ struct cpu_usage *grp_usage;
+
+ adjust_control_window(p);
+
+ if (!grp || !grp->limit || !rq->need_recheck)
+   return 0;
+
+ grp_usage = per_cpu_ptr(grp->cpu_usage, task_cpu(p));
+ if (grp_usage->last_update != rq->last_update) {
```

```
+  /* Replenish tokens */
+  grp_usage->tokens = grp->limit * HZ / 100;
+  grp_usage->last_update = rq->last_update;
+ }
+
+ if (grp_usage->tokens <= 0)
+  return 1;
+
+ return 0;
+}
+
+static inline void rq_set_recheck(struct rq *rq, int check)
+{
+ rq->need_recheck = check;
+}
+
+#else
+
+#define task_starving(p) 0
+
+struct task_grp;
+
+static struct task_grp *task_grp(struct task_struct *p) { return NULL;}
+static void inc_cpu_usage(struct task_struct *p) { }
+static int task_over_cpu_limit(struct task_struct *p) { return 0; }
+static void set_tsk_starving(struct task_struct *p, struct task_grp *grp) { }
+static void clear_tsk_starving(struct task_struct *p, struct task_grp *grp) { }
+static int is_grp_starving(struct task_struct *p) { return 0;}
+static inline void rq_set_recheck(struct rq *rq, int check) { }
+
+#endif  /* CONFIG_CPUMETER */
+
 /*
  * __normal_prio - return the priority that is based on the static
  * priority but is modified by bonuses/penalties.
@@ -847,6 +993,7 @@ static void __activate_task(struct task_
   target = rq->expired;
  enqueue_task(p, target);
  inc_nr_running(p, rq);
+ rq_set_recheck(rq, 1);
 }

 /*
@@ -1586,6 +1733,9 @@ void fastcall sched_fork(struct task_str
  /* Want to start with kernel preemption disabled. */
  task_thread_info(p)->preempt_count = 1;
 #endif
+#ifdef CONFIG_CPUMETER
```

```
+ p->flags &= ~PF_STARVING;
+#endif
  /*
   * Share the timeslice between parent and child, thus the
   * total amount of pending timeslices in the system doesn't change,
@@ -2047,6 +2197,8 @@ static void pull_task(struct rq *src_rq,
 {
  dequeue_task(p, src_array);
  dec_nr_running(p, src_rq);
+ clear_tsk_starving(p, task_grp(p));
+ rq_set_recheck(this_rq, 1);
  set_task_cpu(p, this_cpu);
  inc_nr_running(p, this_rq);
  enqueue_task(p, this_array);
@@ -3068,6 +3220,9 @@ void scheduler_tick(void)
   goto out;
  }
  spin_lock(&rq->lock);
+
+ inc_cpu_usage(p);
+
  /*
   * The task was running during this tick - update the
   * time slice counter. Note: we do not update a thread's
@@ -3094,17 +3249,18 @@ void scheduler_tick(void)
   dequeue_task(p, rq->active);
   set_tsk_need_resched(p);
   p->prio = effective_prio(p);
-  p->time_slice = task_timeslice(p);
   p->first_time_slice = 0;

   if (!rq->expired_timestamp)
    rq->expired_timestamp = jiffies;
-  if (!TASK_INTERACTIVE(p) || expired_starving(rq)) {
+  if (!TASK_INTERACTIVE(p) || expired_starving(rq)
+      || task_over_cpu_limit(p)) {
    enqueue_task(p, rq->expired);
    if (p->static_prio < rq->best_expired_prio)
     rq->best_expired_prio = p->static_prio;
   } else
    enqueue_task(p, rq->active);
+  goto out_unlock;
  } else {
   /*
    * Prevent a too long timeslice allowing a task to monopolize
@@ -3131,6 +3287,14 @@ void scheduler_tick(void)
    set_tsk_need_resched(p);
   }
```

```
  }
+
+ if (task_over_cpu_limit(p)) {
+  dequeue_task(p, rq->active);
+  set_tsk_need_resched(p);
+  enqueue_task(p, rq->expired);
+  set_tsk_starving(p, task_grp(p));
+ }
+
 out_unlock:
  spin_unlock(&rq->lock);
 out:
@@ -3320,7 +3484,7 @@ asmlinkage void __sched schedule(void)
  struct list_head *queue;
  unsigned long long now;
  unsigned long run_time;
- int cpu, idx, new_prio;
+ int cpu, idx, new_prio, array_switch;
  long *switch_count;
  struct rq *rq;

@@ -3379,6 +3543,7 @@ need_resched_nonpreemptible:
   else {
    if (prev->state == TASK_UNINTERRUPTIBLE)
     rq->nr_uninterruptible++;
+   clear_tsk_starving(prev, task_grp(prev));
    deactivate_task(prev, rq);
   }
  }
@@ -3394,11 +3559,15 @@ need_resched_nonpreemptible:
  }

+ array_switch = 0;
+
+pick_next_task:
  array = rq->active;
  if (unlikely(!array->nr_active)) {
   /*
    * Switch the active and expired arrays.
    */
+  array_switch++;
   schedstat_inc(rq, sched_switch);
   rq->active = rq->expired;
   rq->expired = array;
@@ -3411,6 +3580,25 @@ need_resched_nonpreemptible:
  queue = array->queue + idx;
  next = list_entry(queue->next, struct task_struct, run_list);
```

```
+ /* If we have done an array switch twice, it means we cant find any
+  * task which isn't above its limit and hence we just run the
+  * first task on the active array.
+  */
+ if (array_switch < 2 && (task_over_cpu_limit(next) ||
+   (!task_starving(next) && is_grp_starving(next)))) {
+  dequeue_task(next, rq->active);
+  enqueue_task(next, rq->expired);
+  if (next->time_slice)
+   set_tsk_starving(next, task_grp(next));
+  goto pick_next_task;
+ }
+
+ if (task_over_cpu_limit(next))
+  rq_set_recheck(rq, 0);
+ if (!next->time_slice)
+  next->time_slice = task_timeslice(next);
+ clear_tsk_starving(next, task_grp(next));
+
  if (!rt_task(next) && interactive_sleep(next->sleep_type)) {
   unsigned long long delta = now - next->timestamp;
   if (unlikely((long long)(now - next->timestamp) < 0))
@@ -4965,6 +5153,7 @@ static int __migrate_task(struct task_st
  if (!cpu_isset(dest_cpu, p->cpus_allowed))
   goto out;

+ clear_tsk_starving(p, task_grp(p));
  set_task_cpu(p, dest_cpu);
  if (p->array) {
   /*
```

_
--
Regards,
vatsa