Subject: [PATCH 7/9] SMP Load balance changes
Posted by Srivatsa Vaddagiri on Thu, 12 Apr 2007 17:58:44 GMT
View Forum Message <> Reply to Message

When balancing tasks across CPU runqueues, we need to take into account the
quota of task-groups (for example: we dont want to pull all tasks of a group
with 90% limit on the same CPU).

This is easily accomplished by piggy backing on the smpnice mechanism.

This patch also modifies move_tasks() to look at all the arrays of runqueue for
pulling tasks.

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

---

 1 file changed, 119 insertions(+), 36 deletions(-)

```
diff -puN kernel/sched.c~smp kernel/sched.c
--- linux-2.6.20/kernel/sched.c~smp 2007-04-12 09:10:38.000000000 +0530
+++ linux-2.6.20-vatsa/kernel/sched.c 2007-04-12 11:07:15.000000000 +0530
@@ -217,7 +217,7 @@ struct task_grp_rq {
  unsigned long expired_timestamp;
  unsigned short ticks, long_ticks;
    int prio;   /* Priority of the task-group */
- unsigned long last_update;
+ unsigned long last_update, raw_weighted_load;
  struct list_head list;
  struct task_grp *tg;
 };
@@ -925,6 +925,11 @@ static inline int __normal_prio(struct t
 #define RTPRIO_TO_LOAD_WEIGHT(rp) \
  (PRIO_TO_LOAD_WEIGHT(MAX_RT_PRIO) + LOAD_WEIGHT(rp))

+static inline int cpu_quota(struct task_grp *tg)
+{
+ return (tg->ticks * 100) / CPU_CONTROL_SHORT_WINDOW;
+}
+
 static void set_load_weight(struct task_struct *p)
 {
```

```
  if (has_rt_policy(p)) {
@@ -938,21 +943,25 @@ static void set_load_weight(struct task_
    p->load_weight = 0;
   else
 #endif
-   p->load_weight = RTPRIO_TO_LOAD_WEIGHT(p->rt_priority);
+   p->load_weight = (RTPRIO_TO_LOAD_WEIGHT(p->rt_priority)
+       * cpu_quota(task_grp(p))) / 100;
  } else
-  p->load_weight = PRIO_TO_LOAD_WEIGHT(p->static_prio);
+  p->load_weight = (PRIO_TO_LOAD_WEIGHT(p->static_prio)
+       * cpu_quota(task_grp(p))) / 100;
 }

 static inline void
-inc_raw_weighted_load(struct rq *rq, const struct task_struct *p)
+inc_raw_weighted_load(struct rq *rq, struct task_struct *p)
 {
  rq->raw_weighted_load += p->load_weight;
+ task_grp_rq(p)->raw_weighted_load += p->load_weight;
 }

 static inline void
-dec_raw_weighted_load(struct rq *rq, const struct task_struct *p)
+dec_raw_weighted_load(struct rq *rq, struct task_struct *p)
 {
  rq->raw_weighted_load -= p->load_weight;
+ task_grp_rq(p)->raw_weighted_load -= p->load_weight;
 }

 static inline void inc_nr_running(struct task_struct *p, struct rq *rq)
@@ -2295,42 +2304,30 @@ int can_migrate_task(struct task_struct
  return 1;
 }

-#define rq_best_prio(rq) min((rq)->curr->prio, (rq)->expired->best_static_prio)
+#define rq_best_prio(rq) min((rq)->active->best_dyn_prio, \
+      (rq)->expired->best_static_prio)

-/*
- * move_tasks tries to move up to max_nr_move tasks and max_load_move weighted
- * load from busiest to this_rq, as part of a balancing operation within
- * "domain". Returns the number of tasks moved.
- *
- * Called with both runqueues locked.
- */
-static int move_tasks(struct rq *this_rq, int this_cpu, struct rq *busiest,
-        unsigned long max_nr_move, unsigned long max_load_move,
```

```
-       struct sched_domain *sd, enum idle_type idle,
-       int *all_pinned)
+static int __move_tasks(struct task_grp_rq *this_rq, int this_cpu,
+   struct task_grp_rq *busiest, unsigned long max_nr_move,
+   unsigned long max_load_move, struct sched_domain *sd,
+   enum idle_type idle, int *all_pinned, long *load_moved)
 {
  int idx, pulled = 0, pinned = 0, this_best_prio, best_prio,
-     best_prio_seen, skip_for_load;
+     best_prio_seen = 0, skip_for_load;
  struct prio_array *array, *dst_array;
  struct list_head *head, *curr;
  struct task_struct *tmp;
- long rem_load_move;
+ long rem_load_move, grp_load_diff;
+
+ grp_load_diff = busiest->raw_weighted_load - this_rq->raw_weighted_load;
+ rem_load_move = grp_load_diff/2;

- if (max_nr_move == 0 || max_load_move == 0)
+ if (grp_load_diff < 0 || max_nr_move == 0 || max_load_move == 0)
   goto out;

- rem_load_move = max_load_move;
  pinned = 1;
  this_best_prio = rq_best_prio(this_rq);
  best_prio = rq_best_prio(busiest);
- /*
-  * Enable handling of the case where there is more than one task
-  * with the best priority.   If the current running task is one
-  * of those with prio==best_prio we know it won't be moved
-  * and therefore it's safe to override the skip (based on load) of
-  * any task we find with that prio.
-  */
- best_prio_seen = best_prio == busiest->curr->prio;

  /*
   * We first consider expired tasks. Those will likely not be
@@ -2379,7 +2376,7 @@ skip_queue:
  if (skip_for_load && idx < this_best_prio)
   skip_for_load = !best_prio_seen && idx == best_prio;
  if (skip_for_load ||
-     !can_migrate_task(tmp, busiest, this_cpu, sd, idle, &pinned)) {
+     !can_migrate_task(tmp, task_rq(tmp), this_cpu, sd, idle, &pinned)) {

   best_prio_seen |= idx == best_prio;
   if (curr != head)
@@ -2388,7 +2385,8 @@ skip_queue:
```

```
   goto skip_bitmap;
  }

- pull_task(busiest, array, tmp, this_rq, dst_array, this_cpu);
+ pull_task(task_rq(tmp), array, tmp, cpu_rq(this_cpu), dst_array,
+         this_cpu);
  pulled++;
  rem_load_move -= tmp->load_weight;

@@ -2414,9 +2412,98 @@ out:

  if (all_pinned)
   *all_pinned = pinned;
+ *load_moved = grp_load_diff/2 - rem_load_move;
  return pulled;
 }

+static inline int choose_next_array(struct prio_array *arr[], int start_idx)
+{
+ int i;
+
+ for (i = start_idx; arr[i]; i++)
+  if (arr[i]->nr_active)
+   break;
+
+ return i;
+}
+
+/*
+ * move_tasks tries to move up to max_nr_move tasks and max_load_move weighted
+ * load from busiest to this_rq, as part of a balancing operation within
+ * "domain". Returns the number of tasks moved.
+ *
+ * Called with both runqueues locked.
+ */
+static int move_tasks(struct rq *this_rq, int this_cpu, struct rq *busiest,
+       unsigned long max_nr_move, unsigned long max_load_move,
+       struct sched_domain *sd, enum idle_type idle,
+       int *all_pinned)
+{
+ struct task_grp_rq *busy_grp, *this_grp;
+ long load_moved;
+ unsigned long total_nr_moved = 0, nr_moved;
+ int idx;
+ int arr_idx = 0;
+ struct list_head *head, *curr;
+ struct prio_array *array, *array_prefs[5];
+
```

```
+
+ /* order in which tasks are picked up */
+ array_prefs[0] = busiest->greedy_expired;
+ array_prefs[1] = busiest->greedy_active;
+ array_prefs[2] = busiest->expired;
+ array_prefs[3] = busiest->active;
+ array_prefs[4] = NULL;
+
+ arr_idx = choose_next_array(array_prefs, arr_idx);
+ array = array_prefs[arr_idx++];
+
+ if (!array)
+  goto out;
+
+new_array:
+ /* Start searching at priority 0: */
+ idx = 0;
+skip_bitmap:
+ if (!idx)
+  idx = sched_find_first_bit(array->bitmap);
+ else
+  idx = find_next_bit(array->bitmap, MAX_PRIO, idx);
+ if (idx >= MAX_PRIO) {
+  arr_idx = choose_next_array(array_prefs, arr_idx);
+  array = array_prefs[arr_idx++];
+  if (!array)
+   goto out;
+  goto new_array;
+ }
+
+ head = array->queue + idx;
+ curr = head->prev;
+skip_queue:
+ busy_grp = list_entry(curr, struct task_grp_rq, list);
+ this_grp = busy_grp->tg->rq[this_cpu];
+
+ curr = curr->prev;
+
+ nr_moved = __move_tasks(this_grp, this_cpu, busy_grp, max_nr_move,
+   max_load_move, sd, idle, all_pinned, &load_moved);
+
+ total_nr_moved += nr_moved;
+ max_nr_move -= nr_moved;
+ max_load_move -= load_moved;
+
+ if (!max_nr_move || (long)max_load_move <= 0)
+  goto out;
+
```

```
+ if (curr != head)
+  goto skip_queue;
+ idx++;
+ goto skip_bitmap;
+
+out:
+ return total_nr_moved;
+}
+
 /*
  * find_busiest_group finds and returns the busiest CPU group within the
  * domain. It calculates and returns the amount of weighted load which
@@ -7521,11 +7608,6 @@ static int sched_set_quota(struct task_g
  return 0;
 }

-static inline int cpu_quota(struct task_grp *tg)
-{
- return (tg->ticks * 100) / CPU_CONTROL_SHORT_WINDOW;
-}
-
 /* Return assigned quota for this group */
 static int sched_get_quota(struct task_grp *tg)
 {
@@ -7555,6 +7637,7 @@ static void sched_move_task(struct task_
   /* Set tsk->group to tg_old here */
   deactivate_task(tsk, rq);
   /* Set tsk->group to tg_new here */
+  set_load_weight(tsk);
   __activate_task(tsk, rq);
  }


_
--
Regards,
vatsa
```