Subject: [RFC][PATCH 5/6] core changes for group fairness
Posted by Srivatsa Vaddagiri on Mon, 11 Jun 2007 15:56:08 GMT
View Forum Message <> Reply to Message

This patch introduces the core changes in CFS required to accomplish
group fairness at higher levels. It also modifies load balance interface
between classes a bit, so that move_tasks (which is centric to load
balance) can be reused to balance between runqueues of various types
(struct rq in case of SCHED_RT tasks, struct lrq in case of
SCHED_NORMAL/BATCH tasks).

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

---
 include/linux/sched.h | 14 ++
 kernel/sched.c | 155 +++++++++++++++++++----------
 kernel/sched_fair.c | 252 ++-----
 kernel/sched_rt.c | 49 ++++++++-
 4 files changed, 367 insertions(+), 103 deletions(-)

Index: current/include/linux/sched.h
===
--- current.orig/include/linux/sched.h 2007-06-09 15:04:54.000000000 +0530
+++ current/include/linux/sched.h 2007-06-09 15:07:37.000000000 +0530
@@ -866,8 +866,13 @@
  struct task_struct * (*pick_next_task) (struct rq *rq, u64 now);
  void (*put_prev_task) (struct rq *rq, struct task_struct *p, u64 now);

- struct task_struct * (*load_balance_start) (struct rq *rq);
- struct task_struct * (*load_balance_next) (struct rq *rq);
+#ifdef CONFIG_SMP
+ int (*load_balance) (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, unsigned long *total_load_moved);
+#endif
  void (*task_tick) (struct rq *rq, struct task_struct *p);
  void (*task_new) (struct rq *rq, struct task_struct *p);
 };
@@ -893,6 +898,11 @@
  s64 fair_key;
  s64 sum_wait_runtime, sum_sleep_runtime;
  unsigned long wait_runtime_overruns, wait_runtime_underruns;
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ struct sched_entity *parent;
+ struct lrq *lrq,   /* runqueue on which this entity is (to be) queued */
+   *my_q; /* runqueue "owned" by this entity/group */

```
+#endif
 };

 struct task_struct {
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c 2007-06-09 15:07:36.000000000 +0530
+++ current/kernel/sched.c 2007-06-09 15:07:37.000000000 +0530
@@ -133,6 +133,20 @@
  struct rb_root tasks_timeline;
  struct rb_node *rb_leftmost;
  struct rb_node *rb_load_balance_curr;
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ struct sched_entity *curr;
+ struct rq *rq;
+
+ /* leaf lrqs are those that hold tasks (lowest schedulable entity in a
+  * hierarchy). Non-leaf lrqs hold other higher schedulable entities
+  * (like users, containers etc.)
+  *
+  * leaf_lrq_list ties together list of leaf lrq's in a cpu. This list
+  * is used during load balance.
+  */
+ struct list_head leaf_lrq_list;
+#endif
 };

 /*
@@ -161,6 +175,9 @@
 #endif
 #endif
  struct lrq lrq;
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ struct list_head leaf_lrq_list; /* list of leaf lrqs on this cpu */
+#endif

  u64 nr_switches;

@@ -619,6 +636,16 @@
 }

 static void activate_task(struct rq *rq, struct task_struct *p, int wakeup);
+#ifdef CONFIG_SMP
+static int balance_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, unsigned long *load_moved,
```

```
+       int this_best_prio, int best_prio, int best_prio_seen,
+       void *iterator_arg,
+       struct task_struct *(*iterator_start)(void *arg),
+       struct task_struct *(*iterator_next)(void *arg));
+#endif

 #include "sched_stats.h"
 #include "sched_rt.c"
@@ -757,6 +784,9 @@
 static inline void __set_task_cpu(struct task_struct *p, unsigned int cpu)
 {
  task_thread_info(p)->cpu = cpu;
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ p->se.lrq = &cpu_rq(cpu)->lrq;
+#endif
 }

 void set_task_cpu(struct task_struct *p, unsigned int new_cpu)
@@ -781,6 +811,9 @@

  task_thread_info(p)->cpu = new_cpu;

+#ifdef CONFIG_FAIR_GROUP_SCHED
+ p->se.lrq = &new_rq->lrq;
+#endif
 }

 struct migration_req {
@@ -1761,89 +1794,28 @@
  return 1;
 }

-/*
- * Load-balancing iterator: iterate through the hiearchy of scheduling
- * classes, starting with the highest-prio one:
- */
-
-struct task_struct * load_balance_start(struct rq *rq)
-{
- struct sched_class *class = sched_class_highest;
- struct task_struct *p;
-
- do {
-  p = class->load_balance_start(rq);
-  if (p) {
-   rq->load_balance_class = class;
-   return p;
-  }
```

```
-  class = class->next;
- } while (class);
-
- return NULL;
-}
-
-struct task_struct * load_balance_next(struct rq *rq)
-{
- struct sched_class *class = rq->load_balance_class;
- struct task_struct *p;
-
- p = class->load_balance_next(rq);
- if (p)
-  return p;
- /*
-  * Pick up the next class (if any) and attempt to start
-  * the iterator there:
-  */
- while ((class = class->next)) {
-  p = class->load_balance_start(rq);
-  if (p) {
-   rq->load_balance_class = class;
-   return p;
-  }
- }
- return NULL;
-}
-
-#define rq_best_prio(rq) (rq)->curr->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,
+static int balance_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)
+        int *all_pinned, unsigned long *load_moved,
+        int this_best_prio, int best_prio, int best_prio_seen,
+        void *iterator_arg,
+        struct task_struct *(*iterator_start)(void *arg),
+        struct task_struct *(*iterator_next)(void *arg))
 {
```

```
- int pulled = 0, pinned = 0, this_best_prio, best_prio,
-     best_prio_seen, skip_for_load;
+ int pulled = 0, pinned = 0, skip_for_load;
  struct task_struct *p;
- long rem_load_move;
+ long rem_load_move = max_load_move;

  if (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;

  /*
   * Start the load-balancing iterator:
   */
- p = load_balance_start(busiest);
+ p = (*iterator_start)(iterator_arg);
 next:
  if (!p)
   goto out;
@@ -1860,7 +1832,7 @@
      !can_migrate_task(p, busiest, this_cpu, sd, idle, &pinned)) {

   best_prio_seen |= p->prio == best_prio;
-  p = load_balance_next(busiest);
+  p = (*iterator_next)(iterator_arg);
   goto next;
  }

@@ -1875,7 +1847,7 @@
  if (pulled < max_nr_move && rem_load_move > 0) {
   if (p->prio < this_best_prio)
    this_best_prio = p->prio;
-  p = load_balance_next(busiest);
+  p = (*iterator_next)(iterator_arg);
   goto next;
  }
```

```
 out:
@@ -1888,10 +1860,39 @@

  if (all_pinned)
   *all_pinned = pinned;
+ *load_moved = max_load_move - rem_load_move;
  return pulled;
 }

 /*
+ * 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 sched_class *class = sched_class_highest;
+ unsigned long load_moved, total_nr_moved = 0, nr_moved;
+
+ do {
+  nr_moved = class->load_balance(this_rq, this_cpu, busiest,
+     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;
+  class = class->next;
+ } while (class && max_nr_move && max_load_move);
+
+ 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
  * should be moved to restore balance via the imbalance parameter.
@@ -4503,6 +4504,9 @@
 #else
  task_thread_info(idle)->preempt_count = 0;
 #endif
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ idle->se.lrq = &rq->lrq;
+#endif
```

```
 }

 /*
@@ -6086,6 +6090,9 @@
  lrq->nr_running = 0;
  for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
   lrq->cpu_load[j] = 0;
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ lrq->rq = rq;
+#endif
 }

 void __init sched_init(void)
@@ -6110,6 +6117,10 @@
   rq->nr_running = 0;
   rq->clock = 1;
   init_lrq(&rq->lrq, rq);
+#ifdef CONFIG_FAIR_GROUP_SCHED
+  INIT_LIST_HEAD(&rq->leaf_lrq_list);
+  list_add(&rq->lrq.leaf_lrq_list, &rq->leaf_lrq_list);
+#endif

 #ifdef CONFIG_SMP
   for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c 2007-06-09 15:07:36.000000000 +0530
+++ current/kernel/sched_fair.c 2007-06-09 15:07:37.000000000 +0530
@@ -46,6 +46,29 @@
 /*           BEGIN : CFS operations on generic schedulable entities          */
 /*****************************************************************************/

+#ifdef CONFIG_FAIR_GROUP_SCHED
+
+/* cpu runqueue to which this lrq is attached */
+static inline struct rq *lrq_rq(struct lrq *lrq)
+{
+ return lrq->rq;
+}
+
+static inline struct sched_entity *lrq_curr(struct lrq *lrq)
+{
+ return lrq->curr;
+}
+
+/* An entity is a task if it doesn't "own" a runqueue */
+#define entity_is_task(se) (!se->my_q)
+
```

```
+static inline void set_lrq_curr(struct lrq *lrq, struct sched_entity *se)
+{
+ lrq->curr = se;
+}
+
+#else /* CONFIG_FAIR_GROUP_SCHED */
+
 static inline struct rq *lrq_rq(struct lrq *lrq)
 {
  return container_of(lrq, struct rq, lrq);
@@ -69,6 +92,10 @@

 #define entity_is_task(se) 1

+static inline void set_lrq_curr(struct lrq *lrq, struct sched_entity *se) { }
+
+#endif /* CONFIG_FAIR_GROUP_SCHED */
+
 static inline struct task_struct *entity_to_task(struct sched_entity *se)
 {
  return container_of(se, struct task_struct, se);
@@ -119,6 +146,7 @@
  rb_insert_color(&p->run_node, &lrq->tasks_timeline);
  lrq->raw_weighted_load += p->load_weight;
  lrq->nr_running++;
+ p->on_rq = 1;
 }

 static inline void __dequeue_entity(struct lrq *lrq, struct sched_entity *p)
@@ -128,6 +156,7 @@
  rb_erase(&p->run_node, &lrq->tasks_timeline);
  lrq->raw_weighted_load -= p->load_weight;
  lrq->nr_running--;
+ p->on_rq = 0;
 }

 static inline struct rb_node * first_fair(struct lrq *lrq)
@@ -231,6 +260,9 @@

  if (!curr || curtask == rq->idle || !load)
   return;
+
+ BUG_ON(!curr->exec_start);
+
  /*
   * Get the amount of time the current task was running
   * since the last time we changed raw_weighted_load:
@@ -539,6 +571,7 @@
```

```
   */
  update_stats_wait_end(lrq, p, now);
  update_stats_curr_start(lrq, p, now);
+ set_lrq_curr(lrq, p);

  return p;
 }
@@ -557,9 +590,7 @@
    test_tsk_thread_flag(prevtask, TIF_NEED_RESCHED)) {

    dequeue_entity(lrq, prev, 0, now);
-   prev->on_rq = 0;
    enqueue_entity(lrq, prev, 0, now);
-   prev->on_rq = 1;
   } else
    update_curr(lrq, now);
  } else {
@@ -570,6 +601,7 @@

  if (prev->on_rq)
   update_stats_wait_start(lrq, prev, now);
+ set_lrq_curr(lrq, NULL);
 }

 static void update_load_fair(struct lrq *this_lrq)
@@ -640,9 +672,7 @@
   * position within the tree:
   */
  dequeue_entity(lrq, curr, 0, now);
- curr->on_rq = 0;
  enqueue_entity(lrq, curr, 0, now);
- curr->on_rq = 1;

  /*
   * Reschedule if another task tops the current one.
@@ -669,11 +699,70 @@
 /*                      BEGIN : CFS operations on tasks                     */
 /****************************************************************************/

+#ifdef CONFIG_FAIR_GROUP_SCHED
+
+#define for_each_sched_entity(se) \
+  for (; se; se = se->parent)
+
+static inline struct lrq *task_lrq(struct task_struct *p)
+{
+ return p->se.lrq;
+}
```

```
+
+/* runqueue on which this entity is (to be) queued */
+static inline struct lrq *sched_entity_lrq(struct sched_entity *se)
+{
+ return se->lrq;
+}
+
+/* runqueue "owned" by this group */
+static inline struct lrq *group_lrq(struct sched_entity *grp)
+{
+ return grp->my_q;
+}
+
+static inline struct lrq *cpu_lrq(struct lrq *lrq, int this_cpu)
+{
+ return &cpu_rq(this_cpu)->lrq;
+}
+
+#define for_each_leaf_lrq(a, b) \
+  list_for_each_entry(b, &a->leaf_lrq_list, leaf_lrq_list)
+
+#else /* CONFIG_FAIR_GROUP_SCHED */
+
+#define for_each_sched_entity(se) \
+  for (; se; se = NULL)
+
 static inline struct lrq *task_lrq(struct task_struct *p)
 {
  return &task_rq(p)->lrq;
 }

+static inline struct lrq *sched_entity_lrq(struct sched_entity *se)
+{
+ struct task_struct *p = entity_to_task(se);
+ struct rq *rq = task_rq(p);
+
+ return &rq->lrq;
+}
+
+/* runqueue "owned" by this group */
+static inline struct lrq *group_lrq(struct sched_entity *grp)
+{
+ return NULL;
+}
+
+static inline struct lrq *cpu_lrq(struct lrq *lrq, int this_cpu)
+{
+ return &cpu_rq(this_cpu)->lrq;
```

```
+}
+
+#define for_each_leaf_lrq(a, b) \
+  for (b = &a->lrq; b; b = NULL)
+
+#endif /* CONFIG_FAIR_GROUP_SCHED */
+
 /*
  * The enqueue_task method is called before nr_running is
  * increased. Here we update the fair scheduling stats and
@@ -682,10 +771,15 @@
 static void
 enqueue_task_fair(struct rq *rq, struct task_struct *p, int wakeup, u64 now)
 {
- struct lrq *lrq = task_lrq(p);
+ struct lrq *lrq;
  struct sched_entity *se = &p->se;

- enqueue_entity(lrq, se, wakeup, now);
+ for_each_sched_entity(se) {
+  if (se->on_rq)
+   break;
+  lrq = sched_entity_lrq(se);
+  enqueue_entity(lrq, se, wakeup, now);
+ }
 }

 /*
@@ -696,10 +790,16 @@
 static void
 dequeue_task_fair(struct rq *rq, struct task_struct *p, int sleep, u64 now)
 {
- struct lrq *lrq = task_lrq(p);
+ struct lrq *lrq;
  struct sched_entity *se = &p->se;

- dequeue_entity(lrq, se, sleep, now);
+ for_each_sched_entity(se) {
+  lrq = sched_entity_lrq(se);
+  dequeue_entity(lrq, se, sleep, now);
+  /* Don't dequeue parent if it has other entities besides us */
+  if (lrq->raw_weighted_load)
+   break;
+ }
 }

 /*
@@ -721,9 +821,7 @@
```

```
   * position within the tree:
   */
  dequeue_entity(lrq, se, 0, now);
- se->on_rq = 0;
  enqueue_entity(lrq, se, 0, now);
- se->on_rq = 1;

  /*
   * yield-to support: if we are on the same runqueue then
@@ -772,7 +870,10 @@
   if (unlikely(p->policy == SCHED_BATCH))
    granularity = sysctl_sched_batch_wakeup_granularity;

-  __check_preempt_curr_fair(lrq, &p->se, &curr->se, granularity);
+  /* todo: check for preemption at higher levels */
+  if (lrq == task_lrq(p))
+   __check_preempt_curr_fair(lrq, &p->se, &curr->se,
+         granularity);
  }
 }

@@ -781,7 +882,10 @@
  struct lrq *lrq = &rq->lrq;
  struct sched_entity *se;

- se = pick_next_entity(lrq, now);
+ do {
+  se = pick_next_entity(lrq, now);
+  lrq = group_lrq(se);
+ } while (lrq);

  return entity_to_task(se);
 }
@@ -791,19 +895,26 @@
  */
 static void put_prev_task_fair(struct rq *rq, struct task_struct *prev, u64 now)
 {
- struct lrq *lrq = task_lrq(prev);
+ struct lrq *lrq;
  struct sched_entity *se = &prev->se;

  if (prev == rq->idle)
   return;

- put_prev_entity(lrq, se, now);
+ for_each_sched_entity(se) {
+  lrq = sched_entity_lrq(se);
+  put_prev_entity(lrq, se, now);
```

```
+ }
 }

+#ifdef CONFIG_SMP
+
 /**************************************************************/
 /* Fair scheduling class load-balancing methods:
  */

+/* todo: return cache-cold tasks first */
+
 /*
  * Load-balancing iterator. Note: while the runqueue stays locked
  * during the whole iteration, the current task might be
@@ -812,7 +923,7 @@
  * the current task:
  */
 static inline struct task_struct *
-__load_balance_iterator(struct rq *rq, struct rb_node *curr)
+__load_balance_iterator(struct lrq *lrq, struct rb_node *curr)
 {
  struct task_struct *p;

@@ -820,30 +931,121 @@
   return NULL;

  p = rb_entry(curr, struct task_struct, se.run_node);
- rq->lrq.rb_load_balance_curr = rb_next(curr);
+ lrq->rb_load_balance_curr = rb_next(curr);

  return p;
 }

-static struct task_struct * load_balance_start_fair(struct rq *rq)
+static struct task_struct * load_balance_start_fair(void *arg)
 {
- return __load_balance_iterator(rq, first_fair(&rq->lrq));
+ struct lrq *lrq = arg;
+
+ return __load_balance_iterator(lrq, first_fair(lrq));
 }

-static struct task_struct * load_balance_next_fair(struct rq *rq)
+static struct task_struct * load_balance_next_fair(void *arg)
 {
- return __load_balance_iterator(rq, rq->lrq.rb_load_balance_curr);
+ struct lrq *lrq = arg;
+
```

```
+ return __load_balance_iterator(lrq, lrq->rb_load_balance_curr);
 }

+static inline int lrq_best_prio(struct lrq *lrq)
+{
+ struct sched_entity *curr;
+ struct task_struct *p;
+
+ if (!lrq->nr_running)
+  return MAX_PRIO;
+
+ curr = __pick_next_entity(lrq);
+ p = entity_to_task(curr);
+
+ return p->prio;
+}
+
+static int
+load_balance_fair(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, unsigned long *total_load_moved)
+{
+ struct lrq *busy_lrq;
+ unsigned long load_moved, total_nr_moved = 0, nr_moved, rem_load_move;
+
+ rem_load_move = max_load_move;
+
+ for_each_leaf_lrq(busiest, busy_lrq) {
+  struct lrq *this_lrq;
+  long imbalance;
+  unsigned long maxload;
+  int this_best_prio, best_prio, best_prio_seen = 0;
+
+  this_lrq = cpu_lrq(busy_lrq, this_cpu);
+
+  imbalance = busy_lrq->raw_weighted_load -
+       this_lrq->raw_weighted_load;
+  /* Don't pull if this_lrq has more load than busy_lrq */
+  if (imbalance <= 0)
+   continue;
+
+  /* Don't pull more than imbalance/2 */
+  imbalance /= 2;
+  maxload = min(rem_load_move, (unsigned long)imbalance);
+
+  this_best_prio = lrq_best_prio(this_lrq);
+  best_prio = lrq_best_prio(busy_lrq);
```

```
+
+  /*
+   * 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.
+   */
+  if (lrq_curr(busy_lrq) == &busiest->curr->se)
+   best_prio_seen = 1;
+
+  nr_moved = balance_tasks(this_rq, this_cpu, busiest,
+    max_nr_move, maxload, sd, idle, all_pinned,
+    &load_moved, this_best_prio, best_prio,
+    best_prio_seen,
+    /* pass busy_lrq argument into
+     * load_balance_[start|next]_fair iterators
+     */
+    busy_lrq,
+    load_balance_start_fair,
+    load_balance_next_fair);
+
+  total_nr_moved += nr_moved;
+  max_nr_move -= nr_moved;
+  rem_load_move -= load_moved;
+
+  /* todo: break if rem_load_move is < load_per_task */
+  if (!max_nr_move || !rem_load_move)
+   break;
+ }
+
+ *total_load_moved = max_load_move - rem_load_move;
+
+ return total_nr_moved;
+}
+
+#endif /* CONFIG_SMP */
+
 /*
  * scheduler tick hitting a task of our scheduling class:
  */
 static void task_tick_fair(struct rq *rq, struct task_struct *curr)
 {
- struct lrq *lrq = task_lrq(curr);
+ struct lrq *lrq;
  struct sched_entity *se = &curr->se;

- entity_tick(lrq, se);
```

```
+ for_each_sched_entity(se) {
+  lrq = sched_entity_lrq(se);
+  /* todo: reduce tick frequency at higher levels */
+  entity_tick(lrq, se);
+ }
 }

 /*
@@ -880,7 +1082,6 @@
 // p->se.wait_runtime = -(s64)(sysctl_sched_granularity / 2);

  __enqueue_entity(lrq, se);
- p->se.on_rq = 1;
  inc_nr_running(p, rq);
 }

@@ -897,8 +1098,9 @@
  .pick_next_task  = pick_next_task_fair,
  .put_prev_task  = put_prev_task_fair,

- .load_balance_start = load_balance_start_fair,
- .load_balance_next = load_balance_next_fair,
+#ifdef CONFIG_SMP
+ .load_balance  = load_balance_fair,
+#endif
  .task_tick  = task_tick_fair,
  .task_new  = task_new_fair,
 };
Index: current/kernel/sched_rt.c
===================================================================
--- current.orig/kernel/sched_rt.c 2007-06-09 15:04:54.000000000 +0530
+++ current/kernel/sched_rt.c 2007-06-09 15:07:37.000000000 +0530
@@ -100,6 +100,8 @@
  p->se.exec_start = 0;
 }

+#ifdef CONFIG_SMP
+
 /*
  * Load-balancing iterator. Note: while the runqueue stays locked
  * during the whole iteration, the current task might be
@@ -107,8 +109,9 @@
  * achieve that by always pre-iterating before returning
  * the current task:
  */
-static struct task_struct * load_balance_start_rt(struct rq *rq)
+static struct task_struct * load_balance_start_rt(void *arg)
 {
```

```
+ struct rq *rq = (struct rq *)arg;
  struct prio_array *array = &rq->active;
  struct list_head *head, *curr;
  struct task_struct *p;
@@ -132,8 +135,9 @@
  return p;
 }

-static struct task_struct * load_balance_next_rt(struct rq *rq)
+static struct task_struct * load_balance_next_rt(void *arg)
 {
+ struct rq *rq = (struct rq *)arg;
  struct prio_array *array = &rq->active;
  struct list_head *head, *curr;
  struct task_struct *p;
@@ -170,6 +174,42 @@
  return p;
 }

+static int
+load_balance_rt(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, unsigned long *load_moved)
+{
+ int this_best_prio, best_prio, best_prio_seen = 0;
+ int nr_moved;
+
+ best_prio = sched_find_first_bit(busiest->active.bitmap);
+ this_best_prio = sched_find_first_bit(this_rq->active.bitmap);
+
+ /*
+  * 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.
+  */
+ if (busiest->curr->prio == best_prio)
+  best_prio_seen = 1;
+
+ nr_moved = balance_tasks(this_rq, this_cpu, busiest, max_nr_move,
+   max_load_move, sd, idle, all_pinned, load_moved,
+   this_best_prio, best_prio, best_prio_seen,
+   /* pass busiest argument into
+    * load_balance_[start|next]_rt iterators
+    */
+   busiest,
```

```
+   load_balance_start_rt, load_balance_next_rt);
+
+ return nr_moved;
+}
+
+#endif /* CONFIG_SMP */
+
 static void task_tick_rt(struct rq *rq, struct task_struct *p)
 {
  /*
@@ -204,8 +244,9 @@
  .pick_next_task  = pick_next_task_rt,
  .put_prev_task  = put_prev_task_rt,

- .load_balance_start = load_balance_start_rt,
- .load_balance_next = load_balance_next_rt,
+#ifdef CONFIG_SMP
+ .load_balance  = load_balance_rt,
+#endif

  .task_tick  = task_tick_rt,
  .task_new  = task_new_rt,
--
Regards,
vatsa
```