Subject: [RFC][PATCH 4/6] Fix (bad?) interactions between SCHED_RT and SCHED_NORMAL tasks
Posted by Srivatsa Vaddagiri on Mon, 11 Jun 2007 15:55:04 GMT
View Forum Message <> Reply to Message

Currently nr_running and raw_weighted_load fields in runqueue affect
some CFS calculations (like distribute_fair_add, enqueue_sleeper etc).

These fields however are shared between tasks of all classes, which can
potentialy affect those calculations for SCHED_NORMAL tasks. However I
do not know of any bad behaviour caused by not splitting these fields (like
this patch does).

This split is neverthless needed for subsequent patches.

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

---
 kernel/sched.c | 134 ++++++++++++++++++++++++++++++++----------------
 kernel/sched_fair.c | 65 ++++++++++++++++++++++++-
 2 files changed, 128 insertions(+), 71 deletions(-)

Index: current/kernel/sched.c
===
--- current.orig/kernel/sched.c 2007-06-09 15:07:32.000000000 +0530
+++ current/kernel/sched.c 2007-06-09 15:07:36.000000000 +0530
@@ -118,6 +118,7 @@

 /* CFS-related fields in a runqueue */
 struct lrq {
+ long nr_running;
  unsigned long raw_weighted_load;
  #define CPU_LOAD_IDX_MAX 5
  unsigned long cpu_load[CPU_LOAD_IDX_MAX];
@@ -125,6 +126,7 @@

  u64 fair_clock, delta_fair_clock;
  u64 exec_clock, delta_exec_clock;
+ u64 last_tick; /* when did we last smoothen cpu load? */
  s64 wait_runtime;
  unsigned long wait_runtime_overruns, wait_runtime_underruns;

@@ -148,12 +150,18 @@
   * remote CPUs use both these fields when doing load calculation.
   */
  long nr_running;
- struct lrq lrq;

```
+ unsigned long raw_weighted_load;
+#ifdef CONFIG_SMP
+ #define CPU_LOAD_IDX_MAX 5
+ unsigned long cpu_load[CPU_LOAD_IDX_MAX];

  unsigned char idle_at_tick;
 #ifdef CONFIG_NO_HZ
  unsigned char in_nohz_recently;
 #endif
+#endif
+ struct lrq lrq;
+
  u64 nr_switches;

  /*
@@ -589,13 +597,13 @@
 static inline void
 inc_raw_weighted_load(struct rq *rq, const struct task_struct *p)
 {
- rq->lrq.raw_weighted_load += p->se.load_weight;
+ rq->raw_weighted_load += p->se.load_weight;
 }

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

 static inline void inc_nr_running(struct task_struct *p, struct rq *rq)
@@ -741,7 +749,7 @@
 /* Used instead of source_load when we know the type == 0 */
 unsigned long weighted_cpuload(const int cpu)
 {
- return cpu_rq(cpu)->lrq.raw_weighted_load;
+ return cpu_rq(cpu)->raw_weighted_load;
 }

 #ifdef CONFIG_SMP
@@ -876,9 +884,9 @@
  struct rq *rq = cpu_rq(cpu);

  if (type == 0)
-  return rq->lrq.raw_weighted_load;
+  return rq->raw_weighted_load;

- return min(rq->lrq.cpu_load[type-1], rq->lrq.raw_weighted_load);
```

```
+ return min(rq->cpu_load[type-1], rq->raw_weighted_load);
 }

 /*
@@ -890,9 +898,9 @@
  struct rq *rq = cpu_rq(cpu);

  if (type == 0)
-  return rq->lrq.raw_weighted_load;
+  return rq->raw_weighted_load;

- return max(rq->lrq.cpu_load[type-1], rq->lrq.raw_weighted_load);
+ return max(rq->cpu_load[type-1], rq->raw_weighted_load);
 }

 /*
@@ -903,7 +911,7 @@
  struct rq *rq = cpu_rq(cpu);
  unsigned long n = rq->nr_running;

- return n ? rq->lrq.raw_weighted_load / n : SCHED_LOAD_SCALE;
+ return n ? rq->raw_weighted_load / n : SCHED_LOAD_SCALE;
 }

 /*
@@ -1592,54 +1600,6 @@
  return running + uninterruptible;
 }

-static void update_load_fair(struct rq *this_rq)
-{
- unsigned long this_load, fair_delta, exec_delta, idle_delta;
- u64 fair_delta64, exec_delta64, tmp64;
- unsigned int i, scale;
-
- this_rq->lrq.nr_load_updates++;
- if (!(sysctl_sched_features & 64)) {
-  this_load = this_rq->lrq.raw_weighted_load;
-  goto do_avg;
- }
-
- fair_delta64 = this_rq->lrq.delta_fair_clock + 1;
- this_rq->lrq.delta_fair_clock = 0;
-
- exec_delta64 = this_rq->lrq.delta_exec_clock + 1;
- this_rq->lrq.delta_exec_clock = 0;
-
- if (fair_delta64 > (u64)LONG_MAX)
```

```
-  fair_delta64 = (u64)LONG_MAX;
- fair_delta = (unsigned long)fair_delta64;
-
- if (exec_delta64 > (u64)TICK_NSEC)
-  exec_delta64 = (u64)TICK_NSEC;
- exec_delta = (unsigned long)exec_delta64;
-
- idle_delta = TICK_NSEC - exec_delta;
-
- tmp64 = SCHED_LOAD_SCALE * exec_delta64;
- do_div(tmp64, fair_delta);
- tmp64 *= exec_delta64;
- do_div(tmp64, TICK_NSEC);
- this_load = (unsigned long)tmp64;
-
-do_avg:
- /* Update our load: */
- for (i = 0, scale = 1; i < CPU_LOAD_IDX_MAX; i++, scale += scale) {
-  unsigned long old_load, new_load;
-
-  /* scale is effectively 1 << i now, and >> i divides by scale */
-
-  old_load = this_rq->lrq.cpu_load[i];
-  new_load = this_load;
-
-  this_rq->lrq.cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
- }
-}
-
 #ifdef CONFIG_SMP

 /*
@@ -2003,7 +1963,7 @@

    avg_load += load;
    sum_nr_running += rq->nr_running;
-   sum_weighted_load += rq->lrq.raw_weighted_load;
+   sum_weighted_load += rq->raw_weighted_load;
   }

   /*
@@ -2238,11 +2198,11 @@
   rq = cpu_rq(i);

   if (rq->nr_running == 1 &&
-     rq->lrq.raw_weighted_load > imbalance)
+     rq->raw_weighted_load > imbalance)
    continue;
```

```
-  if (rq->lrq.raw_weighted_load > max_load) {
-   max_load = rq->lrq.raw_weighted_load;
+  if (rq->raw_weighted_load > max_load) {
+   max_load = rq->raw_weighted_load;
    busiest = rq;
   }
  }
@@ -2576,6 +2536,32 @@
  spin_unlock(&target_rq->lock);
 }

+static void update_load(struct rq *this_rq)
+{
+ unsigned long this_load;
+ unsigned int i, scale;
+
+ this_load = this_rq->raw_weighted_load;
+
+ /* Update our load: */
+ for (i = 0, scale = 1; i < CPU_LOAD_IDX_MAX; i++, scale += scale) {
+  unsigned long old_load, new_load;
+
+  /* scale is effectively 1 << i now, and >> i divides by scale */
+
+  old_load = this_rq->cpu_load[i];
+  new_load = this_load;
+  /*
+   * Round up the averaging division if load is increasing. This
+   * prevents us from getting stuck on 9 if the load is 10, for
+   * example.
+   */
+  if (new_load > old_load)
+   new_load += scale-1;
+  this_rq->cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
+ }
+}
+
 #ifdef CONFIG_NO_HZ
 static struct {
  atomic_t load_balancer;
@@ -2822,14 +2808,14 @@
  if (time_after_eq(jiffies, rq->next_balance))
   raise_softirq(SCHED_SOFTIRQ);
 }
-#else
+#else /* CONFIG_SMP */
 /*
```

```
 * on UP we do not need to balance between CPUs:
 */
static inline void idle_balance(int cpu, struct rq *rq)
{
}
-#endif
+#endif /* CONFIG_SMP */

DEFINE_PER_CPU(struct kernel_stat, kstat);

@@ -2953,8 +2939,8 @@

 if (!idle_at_tick)
  task_running_tick(rq, p);
- update_load_fair(rq);
 #ifdef CONFIG_SMP
+ update_load(rq);
 rq->idle_at_tick = idle_at_tick;
 trigger_load_balance(cpu);
 #endif
@@ -6090,6 +6076,18 @@
  && addr < (unsigned long)__sched_text_end);
}

+static inline void init_lrq(struct lrq *lrq, struct rq *rq)
+{
+ int j;
+
+ lrq->tasks_timeline = RB_ROOT;
+ lrq->fair_clock = 1;
+ lrq->last_tick = rq_clock(rq);
+ lrq->nr_running = 0;
+ for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
+  lrq->cpu_load[j] = 0;
+}
+
void __init sched_init(void)
{
 int highest_cpu = 0;
@@ -6110,12 +6108,12 @@
  spin_lock_init(&rq->lock);
  lockdep_set_class(&rq->lock, &rq->rq_lock_key);
  rq->nr_running = 0;
-  rq->lrq.tasks_timeline = RB_ROOT;
-  rq->clock = rq->lrq.fair_clock = 1;
+  rq->clock = 1;
+  init_lrq(&rq->lrq, rq);
```

```
-  for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
-   rq->lrq.cpu_load[j] = 0;
 #ifdef CONFIG_SMP
+  for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
+   rq->cpu_load[j] = 0;
   rq->sd = NULL;
   rq->active_balance = 0;
   rq->push_cpu = 0;
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c 2007-06-09 15:07:33.000000000 +0530
+++ current/kernel/sched_fair.c 2007-06-09 15:07:36.000000000 +0530
@@ -64,9 +64,7 @@

 static long lrq_nr_running(struct lrq *lrq)
 {
- struct rq *rq = lrq_rq(lrq);
-
- return rq->nr_running;
+ return lrq->nr_running;
 }

 #define entity_is_task(se) 1
@@ -119,6 +117,8 @@

  rb_link_node(&p->run_node, parent, link);
  rb_insert_color(&p->run_node, &lrq->tasks_timeline);
+ lrq->raw_weighted_load += p->load_weight;
+ lrq->nr_running++;
 }

 static inline void __dequeue_entity(struct lrq *lrq, struct sched_entity *p)
@@ -126,6 +126,8 @@
  if (lrq->rb_leftmost == &p->run_node)
   lrq->rb_leftmost = NULL;
  rb_erase(&p->run_node, &lrq->tasks_timeline);
+ lrq->raw_weighted_load -= p->load_weight;
+ lrq->nr_running--;
 }

 static inline struct rb_node * first_fair(struct lrq *lrq)
@@ -570,12 +572,69 @@
   update_stats_wait_start(lrq, prev, now);
 }

+static void update_load_fair(struct lrq *this_lrq)
+{
+ unsigned long this_load, fair_delta, exec_delta, idle_delta;
```

```
+ u64 fair_delta64, exec_delta64, tmp64;
+ unsigned int i, scale;
+
+ this_lrq->nr_load_updates++;
+ if (!(sysctl_sched_features & 64)) {
+  this_load = this_lrq->raw_weighted_load;
+  goto do_avg;
+ }
+
+ fair_delta64 = this_lrq->delta_fair_clock + 1;
+ this_lrq->delta_fair_clock = 0;
+
+ exec_delta64 = this_lrq->delta_exec_clock + 1;
+ this_lrq->delta_exec_clock = 0;
+
+ if (fair_delta64 > (u64)LONG_MAX)
+  fair_delta64 = (u64)LONG_MAX;
+ fair_delta = (unsigned long)fair_delta64;
+
+ if (exec_delta64 > (u64)TICK_NSEC)
+  exec_delta64 = (u64)TICK_NSEC;
+ exec_delta = (unsigned long)exec_delta64;
+
+ idle_delta = TICK_NSEC - exec_delta;
+
+ tmp64 = SCHED_LOAD_SCALE * exec_delta64;
+ do_div(tmp64, fair_delta);
+ tmp64 *= exec_delta64;
+ do_div(tmp64, TICK_NSEC);
+ this_load = (unsigned long)tmp64;
+
+do_avg:
+ /* Update our load: */
+ for (i = 0, scale = 1; i < CPU_LOAD_IDX_MAX; i++, scale += scale) {
+  unsigned long old_load, new_load;
+
+  /* scale is effectively 1 << i now, and >> i divides by scale */
+
+  old_load = this_lrq->cpu_load[i];
+  new_load = this_load;
+
+  this_lrq->cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
+ }
+}
+
 static void entity_tick(struct lrq *lrq, struct sched_entity *curr)
 {
  struct sched_entity *next;
```

```
  struct rq *rq = lrq_rq(lrq);
  u64 now = __rq_clock(rq);

+ /* replay load smoothening for all ticks we lost */
+ while (time_after_eq64(now, lrq->last_tick)) {
+  update_load_fair(lrq);
+  lrq->last_tick += TICK_NSEC;
+ }
+ /* deal with time wraps */
+ if (unlikely(now - lrq->last_tick > TICK_NSEC))
+  lrq->last_tick = now;
+
  /*
   * Dequeue and enqueue the task to update its
   * position within the tree:
--
Regards,
vatsa
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers