Subject: [PATCH 1/9] split runqueue
Posted by Srivatsa Vaddagiri on Thu, 12 Apr 2007 17:52:47 GMT
View Forum Message <> Reply to Message

This patch introduces a new runqueue structure (struct task_grp_rq) which is
used to hold "related" tasks together. The structure is quite similar to how
the runqueue structure looks today.

The scheme is every task group will have its own per-cpu runqueues (which is
of type 'struct task_grp_rq'), to hold its tasks together on each CPU.

This is in addition to a per-cpu runqueue which holds runnable task-groups
themselves.

The patch also converts over several references of 'struct rq' to
'struct task_grp_rq' (since tasks now reside in task_grp_rq).

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

```
diff -puN init/Kconfig~cpu-controller-v2-split-runqueue init/Kconfig
--- linux-2.6.20/init/Kconfig~cpu-controller-v2-split-runqueue 2007-04-11 11:48:33.000000000
+0530
+++ linux-2.6.20-vatsa/init/Kconfig 2007-04-11 11:48:33.000000000 +0530
@@ -285,6 +285,14 @@ config CONTAINER_CPUACCT
    Provides a simple Resource Controller for monitoring the
    total CPU consumed by the tasks in a container

+config CPUMETER
+ bool "CPU resource control"
+ depends on CPUSETS && EXPERIMENTAL
+ help
+   This options lets you create cpu resource partitions within
+   cpusets. Each resource partition can be given a different quota
+   of CPU usage.
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
diff -puN kernel/sched.c~cpu-controller-v2-split-runqueue kernel/sched.c
--- linux-2.6.20/kernel/sched.c~cpu-controller-v2-split-runqueue 2007-04-11 11:48:33.000000000
+0530
+++ linux-2.6.20-vatsa/kernel/sched.c 2007-04-12 11:07:19.000000000 +0530
@@ -192,11 +192,48 @@ static inline unsigned int task_timeslic

 struct prio_array {
```

```
  unsigned int nr_active;
+ int best_static_prio;
  DECLARE_BITMAP(bitmap, MAX_PRIO+1); /* include 1 bit for delimiter */
  struct list_head queue[MAX_PRIO];
 };

 /*
+ * Each task belong to some task-group. Task-groups and what tasks they contain
+ * are defined by the administrator using some suitable interface.
+ * Administrator can also define the CPU bandwidth provided to each task-group.
+ *
+ * Task-groups are given a certain priority to run on every CPU. Currently
+ * task-group priority on a CPU is defined to be the same as that of
+ * highest-priority runnable task it has on that CPU. Task-groups also
+ * get their own runqueue on every CPU. The main runqueue on each CPU is
+ * used to hold task-groups, rather than tasks.
+ *
+ * Scheduling decision on a CPU is now two-step : first pick highest priority
+ * task-group from the main runqueue and next pick highest priority task from
+ * the runqueue of that group. Both decisions are of O(1) complexity.
+ */
+
+/* runqueue used for every task-group */
+struct task_grp_rq {
+ struct prio_array arrays[2];
+ struct prio_array *active, *expired, *rq_array;
+ unsigned long expired_timestamp;
+ int prio;   /* Priority of the task-group */
+ struct list_head list;
+ struct task_grp *tg;
+};
+
+static DEFINE_PER_CPU(struct task_grp_rq, init_tg_rq);
+
+/* task-group object - maintains information about each task-group */
+struct task_grp {
+ struct task_grp_rq *rq[NR_CPUS]; /* runqueue pointer for every cpu */
+};
+
+/* The "default" task-group */
+struct task_grp init_task_grp;
+
+/*
  * This is the main, per-CPU runqueue data structure.
  *
  * Locking rule: those places that want to lock multiple runqueues
@@ -225,14 +262,15 @@ struct rq {
   */
```

```
  unsigned long nr_uninterruptible;

- unsigned long expired_timestamp;
  /* Cached timestamp set by update_cpu_clock() */
  unsigned long long most_recent_timestamp;
  struct task_struct *curr, *idle;
  unsigned long next_balance;
  struct mm_struct *prev_mm;
+ /* these arrays hold task-groups.
+  * xxx: Avoid this for !CONFIG_CPUMETER?
+  */
  struct prio_array *active, *expired, arrays[2];
- int best_expired_prio;
  atomic_t nr_iowait;

 #ifdef CONFIG_SMP
@@ -248,6 +286,7 @@ struct rq {
 #endif

 #ifdef CONFIG_SCHEDSTATS
+ /* xxx: move these to task-group runqueue where necessary */
  /* latency stats */
  struct sched_info rq_sched_info;

@@ -280,6 +319,13 @@ static inline int cpu_of(struct rq *rq)
 #endif
 }

+#define switch_array(array1, array2) \
+ { \
+   struct prio_array *tmp = array2; \
+   array2 = array1; \
+   array1 = tmp; \
+ }
+
 /*
  * The domain tree (rq->sd) is protected by RCU's quiescent state transition.
  * See detach_destroy_domains: synchronize_sched for details.
@@ -293,6 +339,7 @@ static inline int cpu_of(struct rq *rq)
 #define cpu_rq(cpu)  (&per_cpu(runqueues, (cpu)))
 #define this_rq()  (&__get_cpu_var(runqueues))
 #define task_rq(p)  cpu_rq(task_cpu(p))
+#define task_grp_rq(p)          (task_grp(p)->rq[task_cpu(p)])
 #define cpu_curr(cpu)  (cpu_rq(cpu)->curr)

 #ifndef prepare_arch_switch
@@ -372,6 +419,15 @@ static inline void finish_lock_switch(st
 }
```

```
 #endif /* __ARCH_WANT_UNLOCKED_CTXSW */

+/* return the task-group to which a task belongs */
+static inline struct task_grp *task_grp(struct task_struct *p)
+{
+ /* Simply return the default group for now. A later patch modifies
+  * this function.
+  */
+ return &init_task_grp;
+}
+
 /*
  * __task_rq_lock - lock the runqueue a given task resides on.
  * Must be called interrupts disabled.
@@ -847,10 +903,11 @@ static int effective_prio(struct task_st
  */
 static void __activate_task(struct task_struct *p, struct rq *rq)
 {
- struct prio_array *target = rq->active;
+ struct task_grp_rq *tgrq = task_grp_rq(p);
+ struct prio_array *target = tgrq->active;

  if (batch_task(p))
-  target = rq->expired;
+  target = tgrq->expired;
  enqueue_task(p, target);
  inc_nr_running(p, rq);
 }
@@ -860,7 +917,10 @@ static void __activate_task(struct task_
  */
 static inline void __activate_idle_task(struct task_struct *p, struct rq *rq)
 {
- enqueue_task_head(p, rq->active);
+ struct task_grp_rq *tgrq = task_grp_rq(p);
+ struct prio_array *target = tgrq->active;
+
+ enqueue_task_head(p, target);
  inc_nr_running(p, rq);
 }

@@ -2128,7 +2188,7 @@ int can_migrate_task(struct task_struct
  return 1;
 }

-#define rq_best_prio(rq) min((rq)->curr->prio, (rq)->best_expired_prio)
+#define rq_best_prio(rq) min((rq)->curr->prio, (rq)->expired->best_static_prio)

 /*
```

```
  * move_tasks tries to move up to max_nr_move tasks and max_load_move weighted
@@ -3036,6 +3096,8 @@ unsigned long long current_sched_time(co
  return ns;
 }

+#define nr_tasks(tgrq) (tgrq->active->nr_active + tgrq->expired->nr_active)
+
 /*
  * We place interactive tasks back into the active array, if possible.
  *
@@ -3046,13 +3108,13 @@ unsigned long long current_sched_time(co
  * increasing number of running tasks. We also ignore the interactivity
  * if a better static_prio task has expired:
  */
-static inline int expired_starving(struct rq *rq)
+static inline int expired_starving(struct task_grp_rq *rq)
 {
- if (rq->curr->static_prio > rq->best_expired_prio)
+ if (current->static_prio > rq->expired->best_static_prio)
    return 1;
   if (!STARVATION_LIMIT || !rq->expired_timestamp)
    return 0;
- if (jiffies - rq->expired_timestamp > STARVATION_LIMIT * rq->nr_running)
+ if (jiffies - rq->expired_timestamp > STARVATION_LIMIT * nr_tasks(rq))
    return 1;
   return 0;
 }
@@ -3139,7 +3201,9 @@ void account_steal_time(struct task_stru

 static void task_running_tick(struct rq *rq, struct task_struct *p)
 {
- if (p->array != rq->active) {
+ struct task_grp_rq *tgrq = task_grp_rq(p);
+
+ if (p->array != tgrq->active) {
    /* Task has expired but was not scheduled yet */
    set_tsk_need_resched(p);
    return;
@@ -3163,25 +3227,25 @@ static void task_running_tick(struct rq
     set_tsk_need_resched(p);

     /* put it at the end of the queue: */
-    requeue_task(p, rq->active);
+    requeue_task(p, tgrq->active);
    }
    goto out_unlock;
   }
   if (!--p->time_slice) {
```

```
-  dequeue_task(p, rq->active);
+  dequeue_task(p, tgrq->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)) {
-   enqueue_task(p, rq->expired);
-   if (p->static_prio < rq->best_expired_prio)
-    rq->best_expired_prio = p->static_prio;
+  if (!tgrq->expired_timestamp)
+   tgrq->expired_timestamp = jiffies;
+  if (!TASK_INTERACTIVE(p) || expired_starving(tgrq)) {
+   enqueue_task(p, tgrq->expired);
+   if (p->static_prio < tgrq->expired->best_static_prio)
+    rq->expired->best_static_prio = p->static_prio;
   } else
-   enqueue_task(p, rq->active);
+   enqueue_task(p, tgrq->active);
  } else {
   /*
    * Prevent a too long timeslice allowing a task to monopolize
@@ -3202,9 +3266,9 @@ static void task_running_tick(struct rq
   if (TASK_INTERACTIVE(p) && !((task_timeslice(p) -
    p->time_slice) % TIMESLICE_GRANULARITY(p)) &&
    (p->time_slice >= TIMESLICE_GRANULARITY(p)) &&
-   (p->array == rq->active)) {
+   (p->array == tgrq->active)) {

-   requeue_task(p, rq->active);
+   requeue_task(p, tgrq->active);
    set_tsk_need_resched(p);
   }
  }
@@ -3427,6 +3491,7 @@ asmlinkage void __sched schedule(void)
  int cpu, idx, new_prio;
  long *switch_count;
  struct rq *rq;
+ struct task_grp_rq *next_grp;

  /*
   * Test if we are atomic.  Since do_exit() needs to call into
@@ -3495,23 +3560,36 @@ need_resched_nonpreemptible:
   idle_balance(cpu, rq);
   if (!rq->nr_running) {
```

```
    next = rq->idle;
-   rq->expired_timestamp = 0;
    wake_sleeping_dependent(cpu);
    goto switch_tasks;
   }
  }

+ /* Pick a task group first */
+#ifdef CONFIG_CPUMETER
  array = rq->active;
  if (unlikely(!array->nr_active)) {
+  switch_array(rq->active, rq->expired);
+  array = rq->active;
+ }
+ idx = sched_find_first_bit(array->bitmap);
+ queue = array->queue + idx;
+ next_grp = list_entry(queue->next, struct task_grp_rq, list);
+#else
+ next_grp = init_task_grp.rq[cpu];
+#endif
+
+ /* Pick a task within that group next */
+ array = next_grp->active;
+ if (unlikely(!array->nr_active)) {
   /*
    * Switch the active and expired arrays.
    */
   schedstat_inc(rq, sched_switch);
-  rq->active = rq->expired;
-  rq->expired = array;
-  array = rq->active;
-  rq->expired_timestamp = 0;
-  rq->best_expired_prio = MAX_PRIO;
+  switch_array(next_grp->active, next_grp->expired);
+  array = next_grp->active;
+  next_grp->expired_timestamp = 0;
+  next_grp->expired->best_static_prio = MAX_PRIO;
  }

  idx = sched_find_first_bit(array->bitmap);
@@ -3989,11 +4067,13 @@ void rt_mutex_setprio(struct task_struct
  struct prio_array *array;
  unsigned long flags;
  struct rq *rq;
+ struct task_grp_rq *tgrq;
  int oldprio;

  BUG_ON(prio < 0 || prio > MAX_PRIO);
```

```
  rq = task_rq_lock(p, &flags);
+ tgrq = task_grp_rq(p);

  oldprio = p->prio;
  array = p->array;
@@ -4007,7 +4087,7 @@ void rt_mutex_setprio(struct task_struct
    * in the active array!
    */
   if (rt_task(p))
-   array = rq->active;
+   array = tgrq->active;
   enqueue_task(p, array);
   /*
    * Reschedule if we are currently running on this runqueue and
@@ -4582,7 +4662,8 @@ asmlinkage long sys_sched_getaffinity(pi
 asmlinkage long sys_sched_yield(void)
 {
  struct rq *rq = this_rq_lock();
- struct prio_array *array = current->array, *target = rq->expired;
+ struct task_grp_rq *tgrq = task_grp_rq(current);
+ struct prio_array *array = current->array, *target = tgrq->expired;

  schedstat_inc(rq, yld_cnt);
  /*
@@ -4593,13 +4674,13 @@ asmlinkage long sys_sched_yield(void)
   *  array.)
   */
  if (rt_task(current))
-  target = rq->active;
+  target = tgrq->active;

  if (array->nr_active == 1) {
   schedstat_inc(rq, yld_act_empty);
-  if (!rq->expired->nr_active)
+  if (!tgrq->expired->nr_active)
    schedstat_inc(rq, yld_both_empty);
- } else if (!rq->expired->nr_active)
+ } else if (!tgrq->expired->nr_active)
   schedstat_inc(rq, yld_exp_empty);

  if (array != target) {
@@ -5304,9 +5385,9 @@ static void migrate_dead(unsigned int de
 }

 /* release_task() removes task from tasklist, so we won't find dead tasks. */
-static void migrate_dead_tasks(unsigned int dead_cpu)
+static void migrate_dead_tasks_from_grp(struct task_grp_rq *rq,
```

```
+      unsigned int dead_cpu)
 {
- struct rq *rq = cpu_rq(dead_cpu);
  unsigned int arr, i;

  for (arr = 0; arr < 2; arr++) {
@@ -5319,6 +5400,27 @@ static void migrate_dead_tasks(unsigned
   }
  }
 }
+
+static void migrate_dead_tasks(unsigned int dead_cpu)
+{
+ struct rq *rq = cpu_rq(dead_cpu);
+ unsigned int arr, i;
+
+ for (arr = 0; arr < 2; arr++) {
+  for (i = 0; i < MAX_PRIO; i++) {
+   struct list_head *list = &rq->arrays[arr].queue[i];
+
+   while (!list_empty(list)) {
+    struct task_grp_rq *tgrq =
+     list_entry(list->next,
+       struct task_grp_rq, list);
+
+    migrate_dead_tasks_from_grp(tgrq, dead_cpu);
+    list_del(&tgrq->list);
+   }
+  }
+ }
+}
 #endif /* CONFIG_HOTPLUG_CPU */

 /*
@@ -6900,21 +7002,48 @@ int in_sched_functions(unsigned long add
   && addr < (unsigned long)__sched_text_end);
 }

+static void task_grp_rq_init(struct task_grp_rq *tgrq, struct task_grp *tg)
+{
+ int j, k;
+
+ tgrq->active = tgrq->arrays;
+ tgrq->expired = tgrq->arrays + 1;
+ tgrq->rq_array = NULL;
+ tgrq->expired->best_static_prio = MAX_PRIO;
+ tgrq->active->best_static_prio = MAX_PRIO;
+ tgrq->prio = MAX_PRIO;
```

```
+ tgrq->tg = tg;
+ INIT_LIST_HEAD(&tgrq->list);
+
+ for (j = 0; j < 2; j++) {
+  struct prio_array *array;
+
+  array = tgrq->arrays + j;
+  for (k = 0; k < MAX_PRIO; k++) {
+   INIT_LIST_HEAD(array->queue + k);
+   __clear_bit(k, array->bitmap);
+  }
+  // delimiter for bitsearch
+  __set_bit(MAX_PRIO, array->bitmap);
+ }
+}
+
 void __init sched_init(void)
 {
- int i, j, k;
+ int i, j;

  for_each_possible_cpu(i) {
-  struct prio_array *array;
   struct rq *rq;
+  struct task_grp_rq *tgrq;

   rq = cpu_rq(i);
+  tgrq = init_task_grp.rq[i] = &per_cpu(init_tg_rq, i);
   spin_lock_init(&rq->lock);
+  task_grp_rq_init(tgrq, &init_task_grp);
   lockdep_set_class(&rq->lock, &rq->rq_lock_key);
   rq->nr_running = 0;
   rq->active = rq->arrays;
   rq->expired = rq->arrays + 1;
-  rq->best_expired_prio = MAX_PRIO;

 #ifdef CONFIG_SMP
   rq->sd = NULL;
@@ -6929,6 +7058,9 @@ void __init sched_init(void)
   atomic_set(&rq->nr_iowait, 0);

   for (j = 0; j < 2; j++) {
+   struct prio_array *array;
+   int k;
+
    array = rq->arrays + j;
    for (k = 0; k < MAX_PRIO; k++) {
     INIT_LIST_HEAD(array->queue + k);
```

_
--
Regards,
vatsa

_____

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

_____

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