Subject: [PATCH 6/9] Handle dont care groups
Posted by Srivatsa Vaddagiri on Thu, 12 Apr 2007 17:57:54 GMT
View Forum Message <> Reply to Message

Deal with task-groups whose bandwidth hasnt been explicitly set by the
administrator. Unallocated CPU bandwidth is equally distributed among such
"don't care" groups.

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

 linux-2.6.18-root/include/linux/sched.h | 2

---

 1 file changed, 91 insertions(+), 5 deletions(-)

```
diff -puN kernel/sched.c~dont_care kernel/sched.c
--- linux-2.6.20/kernel/sched.c~dont_care 2007-04-12 09:09:48.000000000 +0530
+++ linux-2.6.20-vatsa/kernel/sched.c 2007-04-12 11:07:15.000000000 +0530
@@ -227,6 +227,12 @@ static DEFINE_PER_CPU(struct task_grp_rq
 /* task-group object - maintains information about each task-group */
 struct task_grp {
  unsigned short ticks, long_ticks; /* bandwidth given to task-group */
+  int left_over_pct;
+  int total_dont_care_grps;
+  int dont_care;          /* Does this group care for its bandwidth ? */
+  struct task_grp *parent;
+  struct list_head dont_care_list;
+  struct list_head list;
  struct task_grp_rq *rq[NR_CPUS]; /* runqueue pointer for every cpu */
 };

@@ -7210,6 +7216,12 @@ void __init sched_init(void)

  init_task_grp.ticks = CPU_CONTROL_SHORT_WINDOW;   /* 100% bandwidth */
  init_task_grp.long_ticks = NUM_LONG_TICKS;
+  init_task_grp.left_over_pct = 100;  /* 100% unallocated bandwidth */
+  init_task_grp.parent = NULL;
+  init_task_grp.total_dont_care_grps = 1; /* init_task_grp itself */
+  init_task_grp.dont_care = 1;
+  INIT_LIST_HEAD(&init_task_grp.dont_care_list);
+ list_add_tail(&init_task_grp.list, &init_task_grp.dont_care_list);
```

```
  for_each_possible_cpu(i) {
   struct rq *rq;
@@ -7382,19 +7394,50 @@ void set_curr_task(int cpu, struct task_

 #ifdef CONFIG_CPUMETER

+/* Distribute left over bandwidth equally to all "dont care" task groups */
+static void recalc_dontcare(struct task_grp *tg_root)
+{
+ int ticks;
+ struct list_head *entry;
+
+ if (!tg_root->total_dont_care_grps)
+  return;
+
+ ticks = ((tg_root->left_over_pct / tg_root->total_dont_care_grps) *
+            CPU_CONTROL_SHORT_WINDOW) / 100;
+
+ list_for_each(entry, &tg_root->dont_care_list) {
+  struct task_grp *tg;
+  int i;
+
+  tg = list_entry(entry, struct task_grp, list);
+  tg->ticks = ticks;
+  for_each_possible_cpu(i)
+   tg->rq[i]->ticks = tg->ticks;
+ }
+}
+
 /* Allocate runqueue structures for the new task-group */
-static int sched_create_group(void)
+static int sched_create_group(struct task_grp *tg_parent)
 {
  struct task_grp *tg;
  struct task_grp_rq *tgrq;
  int i;

+ if (tg_parent->parent)
+  /* We don't support hierarchical CPU res mgmt (yet) */
+  return -EINVAL;
+
  tg = kzalloc(sizeof(*tg), GFP_KERNEL);
  if (!tg)
   return -ENOMEM;

  tg->ticks = CPU_CONTROL_SHORT_WINDOW;
  tg->long_ticks = NUM_LONG_TICKS;
+ tg->parent = tg_parent;
```

```
+ tg->dont_care = 1;
+ tg->left_over_pct = 100;
+ INIT_LIST_HEAD(&tg->dont_care_list);

  for_each_possible_cpu(i) {
   tgrq = kzalloc(sizeof(*tgrq), GFP_KERNEL);
@@ -7404,6 +7447,15 @@ static int sched_create_group(void)
   task_grp_rq_init(tgrq, tg);
  }

+ if (tg->parent) {
+  tg->parent->total_dont_care_grps++;
+  list_add_tail(&tg->list, &tg->parent->dont_care_list);
+  recalc_dontcare(tg->parent);
+ } else {
+  tg->total_dont_care_grps = 1;
+  list_add_tail(&tg->list, &tg->dont_care_list);
+ }
+
  /* A later patch will make 'tg' accessible beyond this function */
  return 0;
 oom:
@@ -7418,6 +7470,16 @@ oom:
 static void sched_destroy_group(struct task_grp *tg)
 {
  int i;
+ struct task_grp *tg_root = tg->parent;
+
+ if (!tg_root)
+  tg_root = tg;
+
+ if (tg->dont_care) {
+  tg_root->total_dont_care_grps--;
+  list_del(&tg->list);
+  recalc_dontcare(tg_root);
+ }

  for_each_possible_cpu(i)
   kfree(tg->rq[i]);
@@ -7428,12 +7490,33 @@ static void sched_destroy_group(struct t
 /* Assign quota to this group */
 static int sched_set_quota(struct task_grp *tg, int quota)
 {
- int i;
+ int i, old_quota = 0;
+ struct task_grp *tg_root = tg->parent;
+
+ if (!tg_root)
```

```
+  tg_root = tg;
+
+ if (!tg->dont_care)
+  old_quota = (tg->ticks * 100) / CPU_CONTROL_SHORT_WINDOW;
+
+ if ((quota - old_quota) > tg_root->left_over_pct)
+  return -EINVAL;
+
+ if (tg->dont_care) {
+  tg->dont_care = 0;
+  tg_root->total_dont_care_grps--;
+  list_del(&tg->list);
+ }

  tg->ticks = (quota * CPU_CONTROL_SHORT_WINDOW) / 100;
+ for_each_possible_cpu(i) {
+    tg->rq[i]->ticks = tg->ticks;
+  tg->rq[i]->long_ticks = tg->long_ticks;
+ }

- for_each_possible_cpu(i)
-  tg->rq[i]->ticks = tg->ticks;
+ /* xxx: needs some locking */
+ tg_root->left_over_pct -= (quota - old_quota);
+ recalc_dontcare(tg_root);

  return 0;
 }
@@ -7446,7 +7529,10 @@ static inline int cpu_quota(struct task_
 /* Return assigned quota for this group */
 static int sched_get_quota(struct task_grp *tg)
 {
- return cpu_quota(tg);
+ if (tg->dont_care)
+  return 0;
+ else
+  return cpu_quota(tg);
 }

 /*

_
--
Regards,
vatsa
```