Subject: Re: [PATCH] Hookup group-scheduler with task container infrastructure
Posted by Srivatsa Vaddagiri on Thu, 13 Sep 2007 12:27:21 GMT
View Forum Message <> Reply to Message

On Wed, Sep 12, 2007 at 06:25:37PM +0200, Dmitry Adamushko wrote:
> > +	kfree(tg);
> > +}
>
> kfree(tg->cfs_rq) && kfree(tg->se) ?

oops, yes!

> > +	if (tsk->sched_class != &fair_sched_class)
> > +		goto done;
>
> this check should be redundant now with sched_can_attach() in place.

well, there is remote possibility that task could have changed policies
between sched_can_attach() and sched_move_task()! In the updated patch
below, I have put some hooks to deal with that scenario elsewhere also
(in rt_mutex_setprio and __setscheduler). Pls let me know if you see
any issues with that.

> > +static void set_se_shares(struct sched_entity *se, unsigned long shares)
> > +{
> > +	struct cfs_rq *cfs_rq = se->cfs_rq;
> > +	struct rq *rq = cfs_rq->rq;
> > +	int on_rq;
> > +
> > +	spin_lock_irq(&rq->lock);
> > +
> > +	on_rq = se->on_rq;
> > +	if (on_rq)
> > +		__dequeue_entity(cfs_rq, se);
> > +
> > +	se->load.weight = shares;
> > +	se->load.inv_weight = div64_64((1ULL<<32), shares);
>
> A bit of nit-picking... are you sure, there is no need in non '__'
> versions of dequeue/enqueu() here (at least, for the sake of

You are right, we need the dequeue/enqueue_entity here, as the sched
entity load is changing in between the two function calls.

How does the updated patch look? (Thanks again for all your review so far!)

--

Add interface to control cpu bandwidth allocation to task-groups.

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

---
 include/linux/container_subsys.h | 6
 init/Kconfig | 9 +
 kernel/sched.c | 344 ++++++++++++++++++++++++++++++++++++++--
 kernel/sched_fair.c | 3
 kernel/sched_idletask.c | 5
 kernel/sched_rt.c | 5
 6 files changed, 355 insertions(+), 17 deletions(-)

Index: current/include/linux/container_subsys.h
===
--- current.orig/include/linux/container_subsys.h
+++ current/include/linux/container_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_container)
 #endif

 /* */
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+SUBSYS(cpu)
+#endif
+
+/* */
Index: current/init/Kconfig
===
--- current.orig/init/Kconfig
+++ current/init/Kconfig
@@ -326,6 +326,15 @@ config RESOURCE_COUNTERS
          infrastructure that works with containers
   depends on CONTAINERS

+config FAIR_GROUP_SCHED
+ bool "Fair group scheduler"
+ depends on EXPERIMENTAL && CONTAINERS
+ help
+   This option enables you to group tasks and control CPU resource
+   allocation to such groups.
+
+   Say N if unsure.
+

```
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -179,6 +179,58 @@ struct load_stat {
  unsigned long delta_fair, delta_exec, delta_stat;
 };

+#ifdef CONFIG_FAIR_GROUP_SCHED
+
+#include <linux/container.h>
+
+struct cfs_rq;
+
+/* task group related information */
+struct task_grp {
+ struct container_subsys_state css;
+ /* schedulable entities of this group on each cpu */
+ struct sched_entity **se;
+ /* runqueue "owned" by this group on each cpu */
+ struct cfs_rq **cfs_rq;
+ unsigned long shares;
+};
+
+/* Default task group's sched entity on each cpu */
+static DEFINE_PER_CPU(struct sched_entity, init_sched_entity);
+/* Default task group's cfs_rq on each cpu */
+static DEFINE_PER_CPU(struct cfs_rq, init_cfs_rq) ____cacheline_aligned_in_smp;
+
+static struct sched_entity *init_sched_entity_p[CONFIG_NR_CPUS];
+static struct cfs_rq *init_cfs_rq_p[CONFIG_NR_CPUS];
+
+/* Default task group.
+ *  Every task in system belong to this group at bootup.
+ */
+static struct task_grp init_task_grp =  {
+     .se     = init_sched_entity_p,
+     .cfs_rq = init_cfs_rq_p,
+     };
+
+/* return group to which a task belongs */
+static inline struct task_grp *task_grp(struct task_struct *p)
+{
+ return container_of(task_subsys_state(p, cpu_subsys_id),
+     struct task_grp, css);
```

```
+}
+
+/* Change a task's cfs_rq and parent entity if it moves across CPUs/groups */
+static inline void set_task_cfs_rq(struct task_struct *p)
+{
+ p->se.cfs_rq = task_grp(p)->cfs_rq[task_cpu(p)];
+ p->se.parent = task_grp(p)->se[task_cpu(p)];
+}
+
+#else
+
+static inline void set_task_cfs_rq(struct task_struct *p) { }
+
+#endif /* CONFIG_FAIR_GROUP_SCHED */
+
 /* CFS-related fields in a runqueue */
 struct cfs_rq {
  struct load_weight load;
@@ -208,6 +260,7 @@ struct cfs_rq {
   * list is used during load balance.
   */
  struct list_head leaf_cfs_rq_list; /* Better name : task_cfs_rq_list? */
+ struct task_grp *tg;    /* group that "owns" this runqueue */
 #endif
 };

@@ -405,18 +458,6 @@ unsigned long long cpu_clock(int cpu)

 EXPORT_SYMBOL_GPL(cpu_clock);

-#ifdef CONFIG_FAIR_GROUP_SCHED
-/* Change a task's ->cfs_rq if it moves across CPUs */
-static inline void set_task_cfs_rq(struct task_struct *p)
-{
- p->se.cfs_rq = &task_rq(p)->cfs;
-}
-#else
-static inline void set_task_cfs_rq(struct task_struct *p)
-{
-}
-#endif
-
 #ifndef prepare_arch_switch
 # define prepare_arch_switch(next) do { } while (0)
 #endif
@@ -3987,8 +4028,11 @@ void rt_mutex_setprio(struct task_struct

  oldprio = p->prio;
```

```
  on_rq = p->se.on_rq;
- if (on_rq)
+ if (on_rq) {
   dequeue_task(rq, p, 0);
+  if (task_running(rq, p))
+   p->sched_class->put_prev_task(rq, p);
+ }

  if (rt_prio(prio))
   p->sched_class = &rt_sched_class;
@@ -4007,6 +4051,7 @@ void rt_mutex_setprio(struct task_struct
   if (task_running(rq, p)) {
    if (p->prio > oldprio)
     resched_task(rq->curr);
+   p->sched_class->set_curr_task(rq);
   } else {
    check_preempt_curr(rq, p);
   }
@@ -4293,8 +4338,11 @@ recheck:
  }
  update_rq_clock(rq);
  on_rq = p->se.on_rq;
- if (on_rq)
+ if (on_rq) {
   deactivate_task(rq, p, 0);
+  if (task_running(rq, p))
+   p->sched_class->put_prev_task(rq, p);
+ }
  oldprio = p->prio;
  __setscheduler(rq, p, policy, param->sched_priority);
  if (on_rq) {
@@ -4307,6 +4355,7 @@ recheck:
   if (task_running(rq, p)) {
    if (p->prio > oldprio)
     resched_task(rq->curr);
+   p->sched_class->set_curr_task(rq);
   } else {
    check_preempt_curr(rq, p);
   }
@@ -6567,7 +6616,25 @@ void __init sched_init(void)
   init_cfs_rq(&rq->cfs, rq);
 #ifdef CONFIG_FAIR_GROUP_SCHED
   INIT_LIST_HEAD(&rq->leaf_cfs_rq_list);
-  list_add(&rq->cfs.leaf_cfs_rq_list, &rq->leaf_cfs_rq_list);
+  {
+   struct cfs_rq *cfs_rq = &per_cpu(init_cfs_rq, i);
+   struct sched_entity *se =
+       &per_cpu(init_sched_entity, i);
```

```
+
+    init_cfs_rq_p[i] = cfs_rq;
+    init_cfs_rq(cfs_rq, rq);
+    cfs_rq->tg = &init_task_grp;
+    list_add(&cfs_rq->leaf_cfs_rq_list,
+         &rq->leaf_cfs_rq_list);
+
+    init_sched_entity_p[i] = se;
+    se->cfs_rq = &rq->cfs;
+    se->my_q = cfs_rq;
+    se->load.weight = NICE_0_LOAD;
+   se->load.inv_weight = div64_64(1ULL<<32, NICE_0_LOAD);
+    se->parent = NULL;
+   }
+  init_task_grp.shares = NICE_0_LOAD;
 #endif
   rq->ls.load_update_last = now;
   rq->ls.load_update_start = now;
@@ -6764,3 +6831,250 @@ void set_curr_task(int cpu, struct task_
 }

 #endif
+
+#ifdef CONFIG_FAIR_GROUP_SCHED
+
+/* return corresponding task_grp object of a container */
+static inline struct task_grp *container_tg(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, cpu_subsys_id),
+       struct task_grp, css);
+}
+
+/* allocate runqueue etc for a new task group */
+static struct container_subsys_state *
+sched_create_group(struct container_subsys *ss, struct container *cont)
+{
+ struct task_grp *tg;
+ struct cfs_rq *cfs_rq;
+ struct sched_entity *se;
+ int i;
+
+ if (!cont->parent) {
+  /* This is early initialization for the top container */
+  init_task_grp.css.container = cont;
+  return &init_task_grp.css;
+ }
+
+ /* we support only 1-level deep hierarchical scheduler atm */
```

```
+ if (cont->parent->parent)
+  return ERR_PTR(-EINVAL);
+
+ tg = kzalloc(sizeof(*tg), GFP_KERNEL);
+ if (!tg)
+  return ERR_PTR(-ENOMEM);
+
+ tg->cfs_rq = kzalloc(sizeof(cfs_rq) * num_possible_cpus(), GFP_KERNEL);
+ if (!tg->cfs_rq)
+  goto err;
+ tg->se = kzalloc(sizeof(se) * num_possible_cpus(), GFP_KERNEL);
+ if (!tg->se)
+  goto err;
+
+ for_each_possible_cpu(i) {
+  struct rq *rq = cpu_rq(i);
+
+  cfs_rq = kmalloc_node(sizeof(struct cfs_rq), GFP_KERNEL,
+        cpu_to_node(i));
+  if (!cfs_rq)
+   goto err;
+
+  se = kmalloc_node(sizeof(struct sched_entity), GFP_KERNEL,
+       cpu_to_node(i));
+  if (!se)
+   goto err;
+
+  memset(cfs_rq, 0, sizeof(struct cfs_rq));
+  memset(se, 0, sizeof(struct sched_entity));
+
+  tg->cfs_rq[i] = cfs_rq;
+  init_cfs_rq(cfs_rq, rq);
+  cfs_rq->tg = tg;
+  list_add_rcu(&cfs_rq->leaf_cfs_rq_list, &rq->leaf_cfs_rq_list);
+
+  tg->se[i] = se;
+  se->cfs_rq = &rq->cfs;
+  se->my_q = cfs_rq;
+  se->load.weight = NICE_0_LOAD;
+  se->load.inv_weight = div64_64(1ULL<<32, NICE_0_LOAD);
+  se->parent = NULL;
+ }
+
+ tg->shares = NICE_0_LOAD;
+
+ /* Bind the container to task_grp object we just created */
+ tg->css.container = cont;
+
```

```
+ return &tg->css;
+
+err:
+ for_each_possible_cpu(i) {
+  if (tg->cfs_rq && tg->cfs_rq[i])
+   kfree(tg->cfs_rq[i]);
+  if (tg->se && tg->se[i])
+   kfree(tg->se[i]);
+ }
+ if (tg->cfs_rq)
+  kfree(tg->cfs_rq);
+ if (tg->se)
+  kfree(tg->se);
+ if (tg)
+  kfree(tg);
+
+ return ERR_PTR(-ENOMEM);
+}
+
+
+/* destroy runqueue etc associated with a task group */
+static void sched_destroy_group(struct container_subsys *ss,
+     struct container *cont)
+{
+ struct task_grp *tg = container_tg(cont);
+ struct cfs_rq *cfs_rq;
+ struct sched_entity *se;
+ int i;
+
+ for_each_possible_cpu(i) {
+  cfs_rq = tg->cfs_rq[i];
+  list_del_rcu(&cfs_rq->leaf_cfs_rq_list);
+ }
+
+ /* wait for possible concurrent references to cfs_rqs complete */
+ synchronize_sched();
+
+ /* now it should be safe to free those cfs_rqs */
+ for_each_possible_cpu(i) {
+  cfs_rq = tg->cfs_rq[i];
+  kfree(cfs_rq);
+
+  se = tg->se[i];
+  kfree(se);
+ }
+
+ kfree(tg->cfs_rq);
+ kfree(tg->se);
```

```
+ kfree(tg);
+}
+
+static int sched_can_attach(struct container_subsys *ss,
+        struct container *cont, struct task_struct *tsk)
+{
+ /* We don't support RT-tasks being in separate groups */
+ if (tsk->sched_class != &fair_sched_class)
+  return -EINVAL;
+
+ return 0;
+}
+
+/* change task's runqueue when it moves between groups */
+static void sched_move_task(struct container_subsys *ss, struct container *cont,
+   struct container *old_cont, struct task_struct *tsk)
+{
+ int on_rq, running;
+ unsigned long flags;
+ struct rq *rq;
+
+ rq = task_rq_lock(tsk, &flags);
+
+ if (tsk->sched_class != &fair_sched_class)
+  goto done;
+
+ update_rq_clock(rq);
+
+ running = task_running(rq, tsk);
+ on_rq = tsk->se.on_rq;
+
+ if (on_rq) {
+  dequeue_task(rq, tsk, 0);
+  if (unlikely(running))
+   tsk->sched_class->put_prev_task(rq, tsk);
+ }
+
+ set_task_cfs_rq(tsk);
+
+ if (on_rq) {
+  enqueue_task(rq, tsk, 0);
+  if (unlikely(running))
+   tsk->sched_class->set_curr_task(rq);
+ }
+
+done:
+ task_rq_unlock(rq, &flags);
+}
```

```
+
+static void set_se_shares(struct sched_entity *se, unsigned long shares)
+{
+ struct cfs_rq *cfs_rq = se->cfs_rq;
+ struct rq *rq = cfs_rq->rq;
+ int on_rq;
+
+ spin_lock_irq(&rq->lock);
+
+ on_rq = se->on_rq;
+ if (on_rq)
+  dequeue_entity(cfs_rq, se, 0);
+
+ se->load.weight = shares;
+ se->load.inv_weight = div64_64((1ULL<<32), shares);
+
+ if (on_rq)
+  enqueue_entity(cfs_rq, se, 0);
+
+ spin_unlock_irq(&rq->lock);
+}
+
+static ssize_t cpu_shares_write(struct container *cont, struct cftype *cftype,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ int i;
+ unsigned long shareval;
+ struct task_grp *tg = container_tg(cont);
+ char buffer[2*sizeof(unsigned long) + 1];
+
+ if (nbytes > 2*sizeof(unsigned long)) /* safety check */
+  return -E2BIG;
+
+ if (copy_from_user(buffer, userbuf, nbytes))
+  return -EFAULT;
+
+ buffer[nbytes] = 0; /* nul-terminate */
+ shareval = simple_strtoul(buffer, NULL, 10);
+
+ tg->shares = shareval;
+ for_each_possible_cpu(i)
+  set_se_shares(tg->se[i], shareval);
+
+ return nbytes;
+}
+
+static u64 cpu_shares_read_uint(struct container *cont, struct cftype *cft)
```

```
+{
+ struct task_grp *tg = container_tg(cont);
+
+ return (u64) tg->shares;
+}
+
+struct cftype cpuctl_share = {
+ .name = "shares",
+ .read_uint = cpu_shares_read_uint,
+ .write = cpu_shares_write,
+};
+
+static int sched_populate(struct container_subsys *ss, struct container *cont)
+{
+ return container_add_file(cont, ss, &cpuctl_share);
+}
+
+struct container_subsys cpu_subsys = {
+ .name = "cpu",
+ .create = sched_create_group,
+ .destroy  = sched_destroy_group,
+ .can_attach = sched_can_attach,
+ .attach = sched_move_task,
+ .populate = sched_populate,
+ .subsys_id = cpu_subsys_id,
+ .early_init = 1,
+};
+
+#endif /* CONFIG_FAIR_GROUP_SCHED */
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -798,8 +798,7 @@ static inline struct cfs_rq *group_cfs_r
  */
 static inline struct cfs_rq *cpu_cfs_rq(struct cfs_rq *cfs_rq, int this_cpu)
 {
- /* A later patch will take group into account */
- return &cpu_rq(this_cpu)->cfs;
+ return cfs_rq->tg->cfs_rq[this_cpu];
 }

 /* Iterate thr' all leaf cfs_rq's on a runqueue */
Index: current/kernel/sched_idletask.c
===================================================================
--- current.orig/kernel/sched_idletask.c
+++ current/kernel/sched_idletask.c
@@ -50,6 +50,10 @@ static void task_tick_idle(struct rq *rq
```

```
 {
 }

+static void set_curr_task_idle(struct rq *rq)
+{
+}
+
 /*
  * Simple, special scheduling class for the per-CPU idle tasks:
  */
@@ -66,6 +70,7 @@ static struct sched_class idle_sched_cla

  .load_balance  = load_balance_idle,

+ .set_curr_task          = set_curr_task_idle,
  .task_tick  = task_tick_idle,
  /* no .task_new for idle tasks */
 };
Index: current/kernel/sched_rt.c
===================================================================
--- current.orig/kernel/sched_rt.c
+++ current/kernel/sched_rt.c
@@ -218,6 +218,10 @@ static void task_tick_rt(struct rq *rq,
  }
 }

+static void set_curr_task_rt(struct rq *rq)
+{
+}
+
 static struct sched_class rt_sched_class __read_mostly = {
  .enqueue_task  = enqueue_task_rt,
  .dequeue_task  = dequeue_task_rt,
@@ -230,5 +234,6 @@ static struct sched_class rt_sched_class

  .load_balance  = load_balance_rt,

+ .set_curr_task          = set_curr_task_rt,
  .task_tick  = task_tick_rt,
 };

--
Regards,
vatsa
```