
Subject: Re: [PATCH] Hookup group-scheduler with task container infrastructure
Posted by [Srivatsa Vaddagiri](#) on Tue, 11 Sep 2007 04:41:45 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, Sep 11, 2007 at 12:28:51AM +0200, Dmitry Adamushko wrote:

```
> > +   rq = task_rq_lock(tsk, &flags);
> > +
>
> I guess, update_rq_clock(rq) should be placed here.
>
> humm... do you really need deactivate/activate_task() here? 'rq' and
> p->se.load.weight stay unchanged so enqueue/dequeue_task() would do a
> job, no?
```

Good catch. Here's the updated patch.

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                   | 312 +++++
kernel/sched_fair.c              | 3
4 files changed, 315 insertions(+), 15 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 inline void set_task_cfs_rq(struct task_struct *p)
-{
-}
-#endif
-
#ifdef prepare_arch_switch
# define prepare_arch_switch(next) do { } while (0)
#endif
@@ -6567,7 +6608,25 @@ 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 +6823,230 @@ 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 = kcalloc_node(sizeof(struct cfs_rq), GFP_KERNEL,
+ cpu_to_node(i));
+ if (!cfs_rq)
+ goto err;
+
+ se = kcalloc_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);
+}
+
+/* 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;
+   unsigned long flags;
+   struct rq *rq;
+
+   rq = task_rq_lock(tsk, &flags);
+   update_rq_clock(rq);
+
+   on_rq = tsk->se.on_rq;
+   if (on_rq)
+     dequeue_task(rq, tsk, 0);
+
+   if (unlikely(rq->curr == tsk) && tsk->sched_class == &fair_sched_class)
+     tsk->sched_class->put_prev_task(rq, tsk);
+
+   set_task_cfs_rq(tsk);
+
+   if (on_rq)
+     enqueue_task(rq, tsk, 0);
+
+   if (unlikely(rq->curr == tsk) && tsk->sched_class == &fair_sched_class)
+     tsk->sched_class->set_curr_task(rq);
+
+   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);
+
+ se->load.weight = shares;
+ se->load.inv_weight = div64_64((1ULL<<32), shares);
+
+ if (on_rq)
+ __enqueue_entity(cfs_rq, se);
+
+ 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);

```


