
Subject: Re: [PATCH] Hookup group-scheduler with task container infrastructure
Posted by [Srivatsa Vaddagiri](#) on Wed, 12 Sep 2007 11:33:47 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, Sep 11, 2007 at 11:53:51AM +0200, Dmitry Adamushko wrote:
> I guess, 'rq->curr == tsk' implies a task was on the 'rq' (before
> dequeuing) in this particular case. What's about a minor optimization
> like below (plus, let's make use of task_running()):

Hi Dmitry,
The modified code looks very good to me (and neater too!). Updated
patch below.

> [btw., real-time task can be also added to a container, right?

I am not sure whether we want to allow that. I have been assuming that
only SCHED_NORMAL tasks will be in containers.

Imagine two SCHED_FIFO tasks being put in two different containers -and-
controlling cpu cycle allocation to both containers. Do we actually timeslice
the two SCHED_FIFO tasks now that are across containers? Also we need a
mechanism to choose a container first to schedule and then the real-time
task in it (similar to how its being done in SCHED_NORMAL case). Which
means we need one rt_rq per container per cpu.

Kirill, do you require real-time tasks to be present in containers?

> I guess, it can be used at least for group-aware load-balancing of
> rt_tasks... otherwise, I guess, cpu-resource controlling is not that
> applicable to, say, SCHED_FIFO :-)

Yes, if we were to allow SCHED_FIFO/RR tasks in containers, then the
load balancer would need to spread rt-tasks of a container across CPUs
to ensure low scheduling latencies for them.

> Anyway, to this goal rt_task should become aware of group-related bits
> of the 'struct sched_entity'... and at the moment, the code below is
> effectively just a 'nop' for them.. right?]

Yes. To emphasise this code doesn't support real-time tasks in a container, I
am returning -ENOTSUP when trying to move a rt-task into a container.

Thanks for all your review!

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                   | 330 ++++++
kernel/sched_fair.c               | 3
4 files changed, 333 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 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)
+{

```



```

+ /* 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);

```

```

+}
+
+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);
+
+ 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);
+
+ 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 */
```

--

Regards,
vatsa

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