Subject: [PATCH 3/5] Cleanup code under CONFIG_FAIR_GROUP_SCHED
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:30:47 GMT
View Forum Message <> Reply to Message

With the view of supporting user-id based fair scheduling (and not just
container-based fair scheduling), this patch renames several functions
and makes them independent of whether they are being used for container
or user-id based fair scheduling.

Also fix a problem reported by KAMEZAWA Hiroyuki (wrt allocating
less-sized array for tg->cfs_rq[] and tf->se[]).

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

---
 include/linux/sched.h | 12 +++
 init/Kconfig | 11 +--
 kernel/sched.c | 172 ++++++++++++++++++++++++++++++++-----------------
 kernel/sched_fair.c | 5 +
 4 files changed, 83 insertions(+), 117 deletions(-)

Index: current/include/linux/sched.h
===
--- current.orig/include/linux/sched.h
+++ current/include/linux/sched.h
@@ -135,6 +135,7 @@ extern unsigned long weighted_cpuload(co

 struct seq_file;
 struct cfs_rq;
+struct task_grp;
 #ifdef CONFIG_SCHED_DEBUG
 extern void proc_sched_show_task(struct task_struct *p, struct seq_file *m);
 extern void proc_sched_set_task(struct task_struct *p);
@@ -1833,6 +1834,17 @@ extern int sched_mc_power_savings, sched

 extern void normalize_rt_tasks(void);

+#ifdef CONFIG_FAIR_GROUP_SCHED
+
+extern struct task_grp init_task_grp;
+
+extern struct task_grp *sched_create_group(void);
+extern void sched_destroy_group(struct task_grp *tg);
+extern void sched_move_task(struct task_struct *tsk);
+extern int sched_group_set_shares(struct task_grp *tg, unsigned long shares);

```
+
+#endif
+
 #ifdef CONFIG_TASK_XACCT
 static inline void add_rchar(struct task_struct *tsk, ssize_t amt)
 {
Index: current/init/Kconfig
===================================================================
--- current.orig/init/Kconfig
+++ current/init/Kconfig
@@ -282,13 +282,12 @@ config CPUSETS
    Say N if unsure.

 config FAIR_GROUP_SCHED
- bool "Fair group scheduler"
- depends on EXPERIMENTAL && CONTAINERS
+ bool "Fair group cpu scheduler"
+ default n
+ depends on EXPERIMENTAL
  help
-   This option enables you to group tasks and control CPU resource
-   allocation to such groups.
-
-   Say N if unsure.
+   This feature lets cpu scheduler recognize task groups and control cpu
+   bandwidth allocation to such task groups.

 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -172,13 +172,10 @@ struct rt_prio_array {

 #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 */
@@ -191,22 +188,28 @@ static DEFINE_PER_CPU(struct sched_entit
```

```
 /* 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];
+static struct sched_entity *init_sched_entity_p[NR_CPUS];
+static struct cfs_rq *init_cfs_rq_p[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,
-     };
+struct task_grp init_task_grp =  {
+     .se     = init_sched_entity_p,
+     .cfs_rq = init_cfs_rq_p,
+      };
+
+#define INIT_TASK_GRP_LOAD NICE_0_LOAD
+static int init_task_grp_load = INIT_TASK_GRP_LOAD;

 /* 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);
+ struct task_grp *tg;
+
+ tg  = &init_task_grp;
+
+ return tg;
 }

 /* Change a task's cfs_rq and parent entity if it moves across CPUs/groups */
@@ -250,6 +253,7 @@ struct cfs_rq {
    */
   struct list_head leaf_cfs_rq_list; /* Better name : task_cfs_rq_list? */
   struct task_grp *tg;    /* group that "owns" this runqueue */
+ struct rcu_head rcu;
 #endif
 };

@@ -6513,11 +6517,12 @@ void __init sched_init(void)
     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->load.weight = init_task_grp_load;
+   se->load.inv_weight =
+     div64_64(1ULL<<32, init_task_grp_load);
     se->parent = NULL;
    }
-  init_task_grp.shares = NICE_0_LOAD;
+  init_task_grp.shares = init_task_grp_load;
 #endif

   for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
@@ -6707,45 +6712,28 @@ void set_curr_task(int cpu, struct task_

 #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 *sched_create_group(void)
 {
  struct task_grp *tg;
  struct cfs_rq *cfs_rq;
  struct sched_entity *se;
+ struct rq *rq;
  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);
+ tg->cfs_rq = kzalloc(sizeof(cfs_rq) * NR_CPUS, GFP_KERNEL);
  if (!tg->cfs_rq)
   goto err;
- tg->se = kzalloc(sizeof(se) * num_possible_cpus(), GFP_KERNEL);
+ tg->se = kzalloc(sizeof(se) * NR_CPUS, GFP_KERNEL);
  if (!tg->se)
   goto err;

  for_each_possible_cpu(i) {
-  struct rq *rq = cpu_rq(i);
+  rq = cpu_rq(i);

   cfs_rq = kmalloc_node(sizeof(struct cfs_rq), GFP_KERNEL,
         cpu_to_node(i));
@@ -6763,7 +6751,6 @@ sched_create_group(struct container_subs
   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;
@@ -6773,12 +6760,15 @@ sched_create_group(struct container_subs
   se->parent = NULL;
  }

- tg->shares = NICE_0_LOAD;
+ for_each_possible_cpu(i) {
+  rq = cpu_rq(i);
+  cfs_rq = tg->cfs_rq[i];
+  list_add_rcu(&cfs_rq->leaf_cfs_rq_list, &rq->leaf_cfs_rq_list);
+ }

- /* Bind the container to task_grp object we just created */
- tg->css.container = cont;
+ tg->shares = NICE_0_LOAD;

- return &tg->css;
+ return tg;

 err:
  for_each_possible_cpu(i) {
@@ -6797,24 +6787,14 @@ err:
  return ERR_PTR(-ENOMEM);
 }

-
```

```
-/* destroy runqueue etc associated with a task group */
-static void sched_destroy_group(struct container_subsys *ss,
-      struct container *cont)
+/* rcu callback to free various structures associated with a task group */
+static void free_sched_group(struct rcu_head *rhp)
 {
- struct task_grp *tg = container_tg(cont);
- struct cfs_rq *cfs_rq;
+ struct cfs_rq *cfs_rq = container_of(rhp, struct cfs_rq, rcu);
+ struct task_grp *tg = cfs_rq->tg;
  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];
@@ -6829,19 +6809,29 @@ static void sched_destroy_group(struct c
  kfree(tg);
 }

-static int sched_can_attach(struct container_subsys *ss,
-        struct container *cont, struct task_struct *tsk)
+/* Destroy runqueue etc associated with a task group */
+void sched_destroy_group(struct task_grp *tg)
 {
- /* We don't support RT-tasks being in separate groups */
- if (tsk->sched_class != &fair_sched_class)
-  return -EINVAL;
+ struct cfs_rq *cfs_rq;
+ int i;

- return 0;
+ for_each_possible_cpu(i) {
+  cfs_rq = tg->cfs_rq[i];
+  list_del_rcu(&cfs_rq->leaf_cfs_rq_list);
+ }
+
+ cfs_rq = tg->cfs_rq[0];
+
+ /* wait for possible concurrent references to cfs_rqs complete */
```

```
+ call_rcu(&cfs_rq->rcu, free_sched_group);
 }

-/* 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)
+/* change task's runqueue when it moves between groups.
+ *  The caller of this function should have put the task in its new group
+ *  by now. This function just updates tsk->se.cfs_rq and tsk->se.parent to
+ *  reflect its new group.
+ */
+void sched_move_task(struct task_struct *tsk)
 {
  int on_rq, running;
  unsigned long flags;
@@ -6896,58 +6886,20 @@ static void set_se_shares(struct sched_e
  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 sched_group_set_shares(struct task_grp *tg, unsigned long shares)
 {
  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;
+ if (tg->shares == shares)
+  return 0;

- buffer[nbytes] = 0; /* nul-terminate */
- shareval = simple_strtoul(buffer, NULL, 10);
+ /* return -EINVAL if the new value is not sane */

- tg->shares = shareval;
+ tg->shares = shares;
  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;
-}
+  set_se_shares(tg->se[i], 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);
+ return 0;
 }

-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 */
+#endif  /* CONFIG_FAIR_GROUP_SCHED */
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -878,7 +878,10 @@ static int cfs_rq_best_prio(struct cfs_r
  if (!cfs_rq->nr_running)
   return MAX_PRIO;

- curr = __pick_next_entity(cfs_rq);
+ curr = cfs_rq->curr;
+ if (!curr)
+  curr = __pick_next_entity(cfs_rq);
+
  p = task_of(curr);
```

```
  return p->prio;
```
--
Regards,
vatsa

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