Subject: [PATCH 0/5] Fair group scheduler - various fixes
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:24:47 GMT
View Forum Message <> Reply to Message

Hello Ingo and Andrew,
 Here are various patches to fair group scheduler code present in
sched-devel and in 2.6.23-rc7-mm1. These patches should apply against
both sched-devel git tree and 2.6.23-rc7-mm1 (as they are both in sync
now).

Pls consider for inclusion after review.

[there is still room for improving group fairness obtained on smp
systems. By sending out these patches, i am hoping more folks can test
and improve that aspect]

Ingo,
 You may want to avoid picking Patch 5/5 as it makes sense only in
-mm tree atm.

--
Regards,
vatsa

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

Subject: [PATCH 1/5] Revert recent removal of set_curr_task()
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:28:26 GMT
View Forum Message <> Reply to Message

On Tue, Sep 18, 2007 at 09:36:30PM +0200, dimm wrote:
> here is a few cleanup/simplification/optimization(s)
> based on the recent modifications in the sched-dev tree.

[refer http://marc.info/?l=linux-kernel&m=119014449807290]

[snip]

> (3) rework enqueue/dequeue_entity() to get rid of
> sched_class::set_curr_task()

Dmitry/Ingo,
 I am sorry for not having reviewed this change properly, but I
think we need to revert this.

Some problems with current sched-devel code introduced by this change:

1. SCHED_NORMAL->SCHED_RT transition of current task
 a. we reset current->se.exec_start to 0 (update_stats_curr_end)
    and not initialize it again. As a result, update_curr_rt()
    can calculate bogus values

 b. (for CONFIG_FAIR_GROUP_SCHED):
  i) higher level entities of current task arent put back
     on their cfs_rq
    ii) their corresponding cfs_rq->curr is not set to NULL
   iii) update_stats_wait_start() not called for higher
     level entities of current task.

     These functions are usually accomplished by put_prev_entity()

2. SCHED_RT->SCHED_NORMAL
 a. (minor) we don't initialize se->prev_sum_exec_runtime
    in enqueue_entity() under if (set_curr) { } block.
    As a result the current task may get preempted
    almost immediately?

 b. (for CONFIG_FAIR_GROUP_SCHED):
     i) higher level entities of current task arent taken out
    of their cfs_rq
    ii) their corresponding cfs_rq->curr is not set
   iii) update_stats_wait_end() and update_stats_curr_start()
    arent called for higher level entities of
    current task.

     These functions are usually accomplished by set_next_entity()

 (similar problems exist while changing groups)

In theory its possible to solve these problems w/o reintroducing
set_curr_task(). I tried doing so, but found it clutters dequeue_entity
and enqueue_entity a lot and makes it less readable. It will duplicate what
put_prev_entity() and set_next_entity() are supposed to do. Moreoever
it is slightly inefficient to do all these in dequeue_entity() if we
consider that dequeue_entity can be called on current task for other
reasons as well (like when it is abt to sleep or change its nice value).

Considering these, IMHO, it's best to re-introduce set_curr_task()
and use put_prev_task/set_curr_task whenever the current task is
changing policies/groups.

Let me know what you think.

--

Revert removal of set_curr_task.
Use put_prev_task/set_curr_task when changing groups/policies

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

---
 include/linux/sched.h | 1
 kernel/sched.c | 34 ++++++++++++++++++++++------
 kernel/sched_fair.c | 68 ++++++++++++++++++++++++++++++++----------------
 kernel/sched_idletask.c | 5 +++
 kernel/sched_rt.c | 8 +++++
 5 files changed, 72 insertions(+), 44 deletions(-)

Index: current/include/linux/sched.h
===
--- current.orig/include/linux/sched.h
+++ current/include/linux/sched.h
@@ -870,6 +870,7 @@ struct sched_class {
    struct sched_domain *sd, enum cpu_idle_type idle,
    int *all_pinned, int *this_best_prio);

+ void (*set_curr_task) (struct rq *rq);
  void (*task_tick) (struct rq *rq, struct task_struct *p);
  void (*task_new) (struct rq *rq, struct task_struct *p);
 };
Index: current/kernel/sched.c
===
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -3916,7 +3916,7 @@ EXPORT_SYMBOL(sleep_on_timeout);
 void rt_mutex_setprio(struct task_struct *p, int prio)
 {
  unsigned long flags;
- int oldprio, on_rq;
+ int oldprio, on_rq, running;
  struct rq *rq;

  BUG_ON(prio < 0 || prio > MAX_PRIO);
@@ -3926,8 +3926,12 @@ void rt_mutex_setprio(struct task_struct

  oldprio = p->prio;
  on_rq = p->se.on_rq;
- if (on_rq)

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

  if (rt_prio(prio))
   p->sched_class = &rt_sched_class;
@@ -3937,13 +3941,15 @@ void rt_mutex_setprio(struct task_struct
  p->prio = prio;

  if (on_rq) {
+  if (running)
+   p->sched_class->set_curr_task(rq);
   enqueue_task(rq, p, 0);
   /*
    * Reschedule if we are currently running on this runqueue and
    * our priority decreased, or if we are not currently running on
    * this runqueue and our priority is higher than the current's
    */
-  if (task_running(rq, p)) {
+  if (running) {
    if (p->prio > oldprio)
     resched_task(rq->curr);
   } else {
@@ -4149,7 +4155,7 @@ __setscheduler(struct rq *rq, struct tas
 int sched_setscheduler(struct task_struct *p, int policy,
          struct sched_param *param)
 {
- int retval, oldprio, oldpolicy = -1, on_rq;
+ int retval, oldprio, oldpolicy = -1, on_rq, running;
  unsigned long flags;
  struct rq *rq;

@@ -4231,20 +4237,26 @@ recheck:
  }
  update_rq_clock(rq);
  on_rq = p->se.on_rq;
- if (on_rq)
+ running = task_running(rq, p);
+ if (on_rq) {
   deactivate_task(rq, p, 0);
+  if (running)
+   p->sched_class->put_prev_task(rq, p);
+ }

  oldprio = p->prio;
```

```
  __setscheduler(rq, p, policy, param->sched_priority);

  if (on_rq) {
+   if (running)
+     p->sched_class->set_curr_task(rq);
    activate_task(rq, p, 0);
    /*
     * Reschedule if we are currently running on this runqueue and
     * our priority decreased, or if we are not currently running on
     * this runqueue and our priority is higher than the current's
     */
-   if (task_running(rq, p)) {
+   if (running) {
     if (p->prio > oldprio)
      resched_task(rq->curr);
    } else {
@@ -6845,13 +6857,19 @@ static void sched_move_task(struct conta
  running = task_running(rq, tsk);
  on_rq = tsk->se.on_rq;

- if (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)
+ if (on_rq) {
+   if (unlikely(running))
+     tsk->sched_class->set_curr_task(rq);
    enqueue_task(rq, tsk, 0);
+ }

 done:
  task_rq_unlock(rq, &flags);
Index: current/kernel/sched_fair.c
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -473,20 +473,9 @@ place_entity(struct cfs_rq *cfs_rq, stru
 }

 static void
-enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se,
-   int wakeup, int set_curr)
```

```
+enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se, int wakeup)
 {
  /*
-  * In case of the 'current'.
-  */
- if (unlikely(set_curr)) {
-  update_stats_curr_start(cfs_rq, se);
-  cfs_rq->curr = se;
-  account_entity_enqueue(cfs_rq, se);
-  return;
- }
-
- /*
   * Update the fair clock.
   */
  update_curr(cfs_rq);
@@ -497,7 +486,8 @@ enqueue_entity(struct cfs_rq *cfs_rq, st
  }

  update_stats_enqueue(cfs_rq, se);
- __enqueue_entity(cfs_rq, se);
+ if (se != cfs_rq->curr)
+  __enqueue_entity(cfs_rq, se);
  account_entity_enqueue(cfs_rq, se);
 }

@@ -517,12 +507,8 @@ dequeue_entity(struct cfs_rq *cfs_rq, st
   }
  }
 #endif
- if (likely(se != cfs_rq->curr))
+ if (se != cfs_rq->curr)
   __dequeue_entity(cfs_rq, se);
- else {
-  update_stats_curr_end(cfs_rq, se);
-  cfs_rq->curr = NULL;
- }
  account_entity_dequeue(cfs_rq, se);
 }

@@ -540,15 +526,20 @@ check_preempt_tick(struct cfs_rq *cfs_rq
   resched_task(rq_of(cfs_rq)->curr);
 }

-static inline void
+static void
 set_next_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
 {
```

```
- /*
-  * Any task has to be enqueued before it get to execute on
-  * a CPU. So account for the time it spent waiting on the
-  * runqueue.
-  */
- update_stats_wait_end(cfs_rq, se);
+ /* 'current' is not kept within the tree. */
+ if (se->on_rq) {
+  /*
+   * Any task has to be enqueued before it get to execute on
+   * a CPU. So account for the time it spent waiting on the
+   * runqueue.
+   */
+  update_stats_wait_end(cfs_rq, se);
+  __dequeue_entity(cfs_rq, se);
+ }
+
  update_stats_curr_start(cfs_rq, se);
  cfs_rq->curr = se;
 #ifdef CONFIG_SCHEDSTATS
@@ -569,10 +560,6 @@ static struct sched_entity *pick_next_en
 {
  struct sched_entity *se = __pick_next_entity(cfs_rq);

- /* 'current' is not kept within the tree. */
- if (se)
-  __dequeue_entity(cfs_rq, se);
-
  set_next_entity(cfs_rq, se);

  return se;
@@ -704,17 +691,12 @@ static void enqueue_task_fair(struct rq
 {
  struct cfs_rq *cfs_rq;
  struct sched_entity *se = &p->se;
- int set_curr = 0;
-
- /* Are we enqueuing the current task? */
- if (unlikely(task_running(rq, p)))
-  set_curr = 1;

  for_each_sched_entity(se) {
   if (se->on_rq)
    break;
   cfs_rq = cfs_rq_of(se);
-  enqueue_entity(cfs_rq, se, wakeup, set_curr);
+  enqueue_entity(cfs_rq, se, wakeup);
  }
```

```
 }

@@ -762,7 +744,7 @@ static void yield_task_fair(struct rq *r
    * position within the tree:
    */
   dequeue_entity(cfs_rq, se, 0);
-  enqueue_entity(cfs_rq, se, 0, 1);
+  enqueue_entity(cfs_rq, se, 0);

   return;
  }
@@ -1005,6 +987,19 @@ static void task_new_fair(struct rq *rq,
  resched_task(rq->curr);
 }

+/* Account for a task changing its policy or group.
+ *
+ * This routine is mostly called to set cfs_rq->curr field when a task
+ * migrates between groups/classes.
+ */
+static void set_curr_task_fair(struct rq *rq)
+{
+ struct sched_entity *se = &rq->curr->se;
+
+ for_each_sched_entity(se)
+  set_next_entity(cfs_rq_of(se), se);
+}
+
 /*
  * All the scheduling class methods:
  */
@@ -1020,6 +1015,7 @@ struct sched_class fair_sched_class __re

   .load_balance  = load_balance_fair,

+ .set_curr_task          = set_curr_task_fair,
   .task_tick  = task_tick_fair,
   .task_new  = task_new_fair,
 };
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 +218,13 @@ static void task_tick_rt(struct rq *rq,
  }
 }

+static void set_curr_task_rt(struct rq *rq)
+{
+ struct task_struct *p = rq->curr;
+
+ p->se.exec_start = rq->clock;
+}
+
 static struct sched_class rt_sched_class __read_mostly = {
  .enqueue_task  = enqueue_task_rt,
  .dequeue_task  = dequeue_task_rt,
@@ -230,5 +237,6 @@ 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

Subject: [PATCH 2/5] Fix minor bug in yield + add more debug o/p
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:29:41 GMT
View Forum Message <> Reply to Message

- Fix a minor bug in yield (seen for CONFIG_FAIR_GROUP_SCHED)
- Print nr_running and load information for cfs_rq in /proc/sched_debug
- Print &rq->cfs statistics as well (usefull for group scheduling)

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

---
 kernel/sched_debug.c | 2 ++
 kernel/sched_fair.c | 3 ++-
 2 files changed, 4 insertions(+), 1 deletion(-)

Index: current/kernel/sched_debug.c
```
===================================================================
--- current.orig/kernel/sched_debug.c
+++ current/kernel/sched_debug.c
@@ -136,6 +136,8 @@ void print_cfs_rq(struct seq_file *m, in
    SPLIT_NS(spread0));
  SEQ_printf(m, "  .%-30s: %ld\n", "spread0",
    cfs_rq->nr_sync_min_vruntime);
+ SEQ_printf(m, "  .%-30s: %ld\n", "nr_running", cfs_rq->nr_running);
+ SEQ_printf(m, "  .%-30s: %ld\n", "load", cfs_rq->load.weight);
 }

 static void print_cpu(struct seq_file *m, int cpu)
```
Index: current/kernel/sched_fair.c
```
===================================================================
--- current.orig/kernel/sched_fair.c
+++ current/kernel/sched_fair.c
@@ -726,7 +726,7 @@ static void dequeue_task_fair(struct rq
  */
 static void yield_task_fair(struct rq *rq)
 {
- struct cfs_rq *cfs_rq = &rq->cfs;
+ struct cfs_rq *cfs_rq = task_cfs_rq(rq->curr);
  struct rb_node **link = &cfs_rq->tasks_timeline.rb_node;
  struct sched_entity *rightmost, *se = &rq->curr->se;
  struct rb_node *parent;
@@ -1025,6 +1025,7 @@ static void print_cfs_stats(struct seq_f
 {
  struct cfs_rq *cfs_rq;

+ print_cfs_rq(m, cpu, &cpu_rq(cpu)->cfs);
```

```
 for_each_leaf_cfs_rq(cpu_rq(cpu), cfs_rq)
  print_cfs_rq(m, cpu, cfs_rq);
}
```

--
Regards,
vatsa

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

---

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

---

Subject: [PATCH 4/5] Add fair-user scheduler
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:35:18 GMT
View Forum Message <> Reply to Message

Enable user-id based fair group scheduling. This is usefull for anyone
who wants to test the group scheduler w/o having to enable
CONFIG_CGROUPS.

A separate scheduling group (i.e struct task_grp) is automatically created for
every new user added to the system. Upon uid change for a task, it is made to
move to the corresponding scheduling group.

A /proc tunable (/proc/root_user_share) is also provided to tune root
user's quota of cpu bandwidth.

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

```
 include/linux/sched.h |    4 +++
 init/Kconfig          |   13 ++++++++++
 kernel/sched.c        |    9 ++++++++
 kernel/sched_debug.c  |   52 ++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
 kernel/user.c         |   43 ++++++++++++++++++++++++++++++++++++++++++++++
 5 files changed, 121 insertions(+)

Index: linux-2.6.23-rc6/include/linux/sched.h
===================================================================
--- linux-2.6.23-rc6.orig/include/linux/sched.h
+++ linux-2.6.23-rc6/include/linux/sched.h
@@ -596,6 +596,10 @@ struct user_struct {
  /* Hash table maintenance information */
  struct hlist_node uidhash_node;
  uid_t uid;
+
+#ifdef CONFIG_FAIR_USER_SCHED
+ struct task_grp *tg;
+#endif
 };

 extern struct user_struct *find_user(uid_t);
Index: linux-2.6.23-rc6/init/Kconfig
===================================================================
--- linux-2.6.23-rc6.orig/init/Kconfig
+++ linux-2.6.23-rc6/init/Kconfig
@@ -289,6 +289,19 @@ config FAIR_GROUP_SCHED
    This feature lets cpu scheduler recognize task groups and control cpu
    bandwidth allocation to such task groups.

+choice
+ depends on FAIR_GROUP_SCHED
+ prompt "Basis for grouping tasks"
+ default FAIR_USER_SCHED
+
+  config FAIR_USER_SCHED
+   bool "user id"
+   help
+     This option will choose userid as the basis for grouping
+    tasks, thus providing equal cpu bandwidth to each user.
+
+endchoice
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
Index: linux-2.6.23-rc6/kernel/sched.c
===================================================================
--- linux-2.6.23-rc6.orig/kernel/sched.c
+++ linux-2.6.23-rc6/kernel/sched.c
@@ -199,7 +199,12 @@ struct task_grp init_task_grp =  {
     .cfs_rq = init_cfs_rq_p,
```

```
     };

+#ifdef CONFIG_FAIR_USER_SCHED
+#define INIT_TASK_GRP_LOAD 2*NICE_0_LOAD
+#else
 #define INIT_TASK_GRP_LOAD NICE_0_LOAD
+#endif
+
 static int init_task_grp_load = INIT_TASK_GRP_LOAD;

 /* return group to which a task belongs */
@@ -207,7 +212,11 @@ static inline struct task_grp *task_grp(
 {
  struct task_grp *tg;

+#ifdef CONFIG_FAIR_USER_SCHED
+ tg = p->user->tg;
+#else
  tg  = &init_task_grp;
+#endif

  return tg;
 }
Index: linux-2.6.23-rc6/kernel/sched_debug.c
===================================================================
--- linux-2.6.23-rc6.orig/kernel/sched_debug.c
+++ linux-2.6.23-rc6/kernel/sched_debug.c
@@ -214,6 +214,49 @@ static void sysrq_sched_debug_show(void)
  sched_debug_show(NULL, NULL);
 }

+#ifdef CONFIG_FAIR_USER_SCHED
+
+static DEFINE_MUTEX(root_user_share_mutex);
+
+static int
+root_user_share_read_proc(char *page, char **start, off_t off, int count,
+     int *eof, void *data)
+{
+ int len;
+
+ len = sprintf(page, "%d\n", init_task_grp_load);
+
+ return len;
+}
+
+static int
+root_user_share_write_proc(struct file *file, const char __user *buffer,
```

```
+     unsigned long count, void *data)
+{
+ unsigned long shares;
+ char kbuf[sizeof(unsigned long)+1];
+ int rc = 0;
+
+ if (copy_from_user(kbuf, buffer, sizeof(kbuf)))
+  return -EFAULT;
+
+ shares = simple_strtoul(kbuf, NULL, 0);
+
+ if (!shares)
+  shares = NICE_0_LOAD;
+
+ mutex_lock(&root_user_share_mutex);
+
+ init_task_grp_load = shares;
+ rc = sched_group_set_shares(&init_task_grp, shares);
+
+ mutex_unlock(&root_user_share_mutex);
+
+ return (rc < 0 ? rc : count);
+}
+
+#endif /* CONFIG_FAIR_USER_SCHED */
+
 static int sched_debug_open(struct inode *inode, struct file *filp)
 {
  return single_open(filp, sched_debug_show, NULL);
@@ -236,6 +279,15 @@ static int __init init_sched_debug_procf

  pe->proc_fops = &sched_debug_fops;

+#ifdef CONFIG_FAIR_USER_SCHED
+ pe = create_proc_entry("root_user_share", 0644, NULL);
+ if (!pe)
+  return -ENOMEM;
+
+ pe->read_proc = root_user_share_read_proc;
+ pe->write_proc = root_user_share_write_proc;
+#endif
+
  return 0;
 }

Index: linux-2.6.23-rc6/kernel/user.c
===================================================================
--- linux-2.6.23-rc6.orig/kernel/user.c
```

```
+++ linux-2.6.23-rc6/kernel/user.c
@@ -50,8 +50,41 @@ struct user_struct root_user = {
  .uid_keyring = &root_user_keyring,
  .session_keyring = &root_session_keyring,
 #endif
+#ifdef CONFIG_FAIR_USER_SCHED
+ .tg  = &init_task_grp,
+#endif
 };

+#ifdef CONFIG_FAIR_USER_SCHED
+static void sched_destroy_user(struct user_struct *up)
+{
+ sched_destroy_group(up->tg);
+}
+
+static int sched_create_user(struct user_struct *up)
+{
+ int rc = 0;
+
+ up->tg = sched_create_group();
+ if (IS_ERR(up->tg))
+  rc = -ENOMEM;
+
+ return rc;
+}
+
+static void sched_switch_user(struct task_struct *p)
+{
+ sched_move_task(p);
+}
+
+#else /* CONFIG_FAIR_USER_SCHED */
+
+static void sched_destroy_user(struct user_struct *up) { }
+static int sched_create_user(struct user_struct *up) { return 0; }
+static void sched_switch_user(struct task_struct *p) { }
+
+#endif /* CONFIG_FAIR_USER_SCHED */
+
 /*
  * These routines must be called with the uidhash spinlock held!
  */
@@ -109,6 +142,7 @@ void free_uid(struct user_struct *up)
  if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
   uid_hash_remove(up);
   spin_unlock_irqrestore(&uidhash_lock, flags);
+  sched_destroy_user(up);
```

```
   key_put(up->uid_keyring);
   key_put(up->session_keyring);
   kmem_cache_free(uid_cachep, up);
@@ -150,6 +184,13 @@ struct user_struct * alloc_uid(struct us
    return NULL;
   }

+  if (sched_create_user(new) < 0) {
+   key_put(new->uid_keyring);
+   key_put(new->session_keyring);
+   kmem_cache_free(uid_cachep, new);
+   return NULL;
+  }
+
   /*
    * Before adding this, check whether we raced
    * on adding the same user already..
@@ -157,6 +198,7 @@ struct user_struct * alloc_uid(struct us
   spin_lock_irq(&uidhash_lock);
   up = uid_hash_find(uid, hashent);
   if (up) {
+   sched_destroy_user(new);
    key_put(new->uid_keyring);
    key_put(new->session_keyring);
    kmem_cache_free(uid_cachep, new);
@@ -184,6 +226,7 @@ void switch_uid(struct user_struct *new_
  atomic_dec(&old_user->processes);
  switch_uid_keyring(new_user);
  current->user = new_user;
+ sched_switch_user(current);

  /*
   * We need to synchronize with __sigqueue_alloc()

--
Regards,
vatsa
```

Subject: Re: [PATCH 1/5] Revert recent removal of set_curr_task()
Posted by Ingo Molnar on Mon, 24 Sep 2007 16:35:25 GMT
View Forum Message <> Reply to Message

* Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com> wrote:

> > (3) rework enqueue/dequeue_entity() to get rid of
> > sched_class::set_curr_task()
>
> Dmitry/Ingo,
> I am sorry for not having reviewed this change properly, but I
> think we need to revert this.

ah, i was wondering about that already. We can certainly skip that
optimization.

> In theory its possible to solve these problems w/o reintroducing
> set_curr_task(). I tried doing so, but found it clutters
> dequeue_entity and enqueue_entity a lot and makes it less readable. It
> will duplicate what put_prev_entity() and set_next_entity() are
> supposed to do. Moreoever it is slightly inefficient to do all these
> in dequeue_entity() if we consider that dequeue_entity can be called
> on current task for other reasons as well (like when it is abt to
> sleep or change its nice value).

yeah, it's not worth it. I'd go for keeping the code unified even if
adds a few instructions runtime overhead, as i'd expect most distros to
enable fair-group-scheduling by default in the future. (once all the
containers infrastructure and tools has trickled down to them)

 Ingo

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

---

Subject: [PATCH 5/5] Add fair "control groups" scheduler
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 16:37:14 GMT
View Forum Message <> Reply to Message

Enable "cgroup" (formerly containers) based fair group scheduling.
This will let administrator create arbitrary groups of tasks (using
"cgroup" psuedo filesystem) and control their cpu bandwidth usage.

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

---
 include/linux/cgroup_subsys.h | 6 ++
 init/Kconfig | 24 +++++---
 kernel/sched.c | 122 ++

```
 3 files changed, 145 insertions(+), 7 deletions(-)

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

 /* */
+
+#ifdef CONFIG_FAIR_CGROUP_SCHED
+SUBSYS(cpu_cgroup)
+#endif
+
+/* */
Index: current/init/Kconfig
===================================================================
--- current.orig/init/Kconfig
+++ current/init/Kconfig
@@ -327,13 +327,6 @@ config FAIR_GROUP_SCHED
    This feature lets cpu scheduler recognize task groups and control cpu
    bandwidth allocation to such task groups.

-config RESOURCE_COUNTERS
- bool "Resource counters"
- help
-   This option enables controller independent resource accounting
-         infrastructure that works with cgroups
- depends on CGROUPS
-
 choice
  depends on FAIR_GROUP_SCHED
  prompt "Basis for grouping tasks"
@@ -345,8 +338,25 @@ choice
     This option will choose userid as the basis for grouping
     tasks, thus providing equal cpu bandwidth to each user.

+ config FAIR_CGROUP_SCHED
+  bool "Control groups"
+  depends on CGROUPS
+  help
+    This option allows you to create arbitrary task groups
+    using the "cgroup" psuedo filesystem and control
+    the cpu bandwidth allocated to each such task group.
+    Refer to Documentation/cgroups.txt for more information
+    on "cgroup" psuedo filesystem.
+
```

```
 endchoice

+config RESOURCE_COUNTERS
+ bool "Resource counters"
+ help
+   This option enables controller independent resource accounting
+         infrastructure that works with cgroups
+ depends on CGROUPS
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
Index: current/kernel/sched.c
===================================================================
--- current.orig/kernel/sched.c
+++ current/kernel/sched.c
@@ -179,10 +179,16 @@ EXPORT_SYMBOL_GPL(cpu_clock);

 #ifdef CONFIG_FAIR_GROUP_SCHED

+#include <linux/cgroup.h>
+
 struct cfs_rq;

 /* task group related information */
 struct task_grp {
+#ifdef CONFIG_FAIR_CGROUP_SCHED
+ struct cgroup_subsys_state css;
+#endif
+
  /* schedulable entities of this group on each cpu */
  struct sched_entity **se;
  /* runqueue "owned" by this group on each cpu */
@@ -221,6 +227,9 @@ static inline struct task_grp *task_grp(

 #ifdef CONFIG_FAIR_USER_SCHED
  tg = p->user->tg;
+#elif CONFIG_FAIR_CGROUP_SCHED
+ tg = container_of(task_subsys_state(p, cpu_cgroup_subsys_id),
+     struct task_grp, css);
 #else
  tg  = &init_task_grp;
 #endif
@@ -6950,3 +6959,116 @@ int sched_group_set_shares(struct task_g
 }

 #endif  /* CONFIG_FAIR_GROUP_SCHED */
+
```

```
+#ifdef CONFIG_FAIR_CGROUP_SCHED
+
+/* return corresponding task_grp object of a cgroup */
+static inline struct task_grp *cgroup_tg(struct cgroup *cont)
+{
+ return container_of(cgroup_subsys_state(cont, cpu_cgroup_subsys_id),
+       struct task_grp, css);
+}
+
+static struct cgroup_subsys_state *
+cpu_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct task_grp *tg;
+
+ if (!cont->parent) {
+  /* This is early initialization for the top cgroup */
+  init_task_grp.css.cgroup = cont;
+  return &init_task_grp.css;
+ }
+
+ /* we support only 1-level deep hierarchical scheduler atm */
+ if (cont->parent->parent)
+  return ERR_PTR(-EINVAL);
+
+ tg = sched_create_group();
+ if (IS_ERR(tg))
+  return ERR_PTR(-ENOMEM);
+
+ /* Bind the cgroup to task_grp object we just created */
+ tg->css.cgroup = cont;
+
+ return &tg->css;
+}
+
+static void cpu_cgroup_destroy(struct cgroup_subsys *ss,
+      struct cgroup *cont)
+{
+ struct task_grp *tg = cgroup_tg(cont);
+
+ sched_destroy_group(tg);
+}
+
+static int cpu_cgroup_can_attach(struct cgroup_subsys *ss,
+         struct cgroup *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;
+}
+
+static void
+cpu_cgroup_attach(struct cgroup_subsys *ss, struct cgroup *cont,
+   struct cgroup *old_cont, struct task_struct *tsk)
+{
+ sched_move_task(tsk);
+}
+
+static ssize_t cpu_shares_write(struct cgroup *cont, struct cftype *cftype,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ unsigned long shareval;
+ struct task_grp *tg = cgroup_tg(cont);
+ char buffer[2*sizeof(unsigned long) + 1];
+ int rc;
+
+ 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);
+
+ rc = sched_group_set_shares(tg, shareval);
+
+ return (rc < 0 ? rc : nbytes);
+}
+
+static u64 cpu_shares_read_uint(struct cgroup *cont, struct cftype *cft)
+{
+ struct task_grp *tg = cgroup_tg(cont);
+
+ return (u64) tg->shares;
+}
+
+static struct cftype cpu_shares = {
+ .name = "shares",
+ .read_uint = cpu_shares_read_uint,
+ .write = cpu_shares_write,
+};
+
+static int cpu_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
+{
+ return cgroup_add_file(cont, ss, &cpu_shares);
+}
+
+struct cgroup_subsys cpu_cgroup_subsys = {
+ .name = "cpu",
+ .create = cpu_cgroup_create,
+ .destroy  = cpu_cgroup_destroy,
+ .can_attach = cpu_cgroup_can_attach,
+ .attach = cpu_cgroup_attach,
+ .populate = cpu_cgroup_populate,
+ .subsys_id = cpu_cgroup_subsys_id,
+ .early_init = 1,
+};
+
+#endif /* CONFIG_FAIR_CGROUP_SCHED */

--
Regards,
vatsa
```

Subject: Re: [PATCH 3/5] Cleanup code under CONFIG_FAIR_GROUP_SCHED
Posted by Randy Dunlap on Mon, 24 Sep 2007 16:53:44 GMT
View Forum Message <> Reply to Message

On Mon, 24 Sep 2007 22:09:37 +0530 Srivatsa Vaddagiri wrote:

> 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/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"

Can we have "CPU" instead of "cpu" ?

> + 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"

~Randy
Phaedrus says that Quality is about caring.

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

---

Subject: Re: [PATCH 4/5] Add fair-user scheduler
Posted by Randy Dunlap on Mon, 24 Sep 2007 16:56:41 GMT
View Forum Message <> Reply to Message

On Mon, 24 Sep 2007 22:10:59 +0530 Srivatsa Vaddagiri wrote:

> ---
> include/linux/sched.h | 4 +++
> init/Kconfig | 13 ++++++++++++
> kernel/sched.c | 9 ++++++++
> kernel/sched_debug.c | 52 +++
> kernel/user.c | 43 +++
> 5 files changed, 121 insertions(+)
>
> Index: linux-2.6.23-rc6/init/Kconfig
> ===
> --- linux-2.6.23-rc6.orig/init/Kconfig
> +++ linux-2.6.23-rc6/init/Kconfig
> @@ -289,6 +289,19 @@ config FAIR_GROUP_SCHED
>   This feature lets cpu scheduler recognize task groups and control cpu
>   bandwidth allocation to such task groups.
>
> +choice
> + depends on FAIR_GROUP_SCHED
> + prompt "Basis for grouping tasks"
> + default FAIR_USER_SCHED
> +
> + config FAIR_USER_SCHED
> +   bool "user id"
> +   help
> +   This option will choose userid as the basis for grouping
> +   tasks, thus providing equal cpu bandwidth to each user.

s/cpu/CPU/g please.

and the "bool", "help" and help text are indented by one tab too much.

> +
> +endchoice
> +
> config SYSFS_DEPRECATED
>   bool "Create deprecated sysfs files"
>   default y

~Randy
Phaedrus says that Quality is about caring.

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

Subject: Re: [PATCH 5/5] Add fair "control groups" scheduler
Posted by Randy Dunlap on Mon, 24 Sep 2007 16:58:15 GMT
View Forum Message <> Reply to Message

On Mon, 24 Sep 2007 22:11:59 +0530 Srivatsa Vaddagiri wrote:

> include/linux/cgroup_subsys.h | 6 ++
> init/Kconfig | 24 +++++---
> kernel/sched.c | 122 ++
> 3 files changed, 145 insertions(+), 7 deletions(-)
>
> Index: current/init/Kconfig
> ===
> --- current.orig/init/Kconfig
> +++ current/init/Kconfig
> @@ -327,13 +327,6 @@ config FAIR_GROUP_SCHED
> This feature lets cpu scheduler recognize task groups and control cpu
> bandwidth allocation to such task groups.
>
> -config RESOURCE_COUNTERS
> - bool "Resource counters"
> - help
> - This option enables controller independent resource accounting
> - infrastructure that works with cgroups
> - depends on CGROUPS
> -
> choice
> depends on FAIR_GROUP_SCHED
> prompt "Basis for grouping tasks"
> @@ -345,8 +338,25 @@ choice
> This option will choose userid as the basis for grouping
> tasks, thus providing equal cpu bandwidth to each user.
>
> + config FAIR_CGROUP_SCHED
> + bool "Control groups"
> + depends on CGROUPS
> + help
> + This option allows you to create arbitrary task groups
> + using the "cgroup" psuedo filesystem and control
> + the cpu bandwidth allocated to each such task group.
> + Refer to Documentation/cgroups.txt for more information
> + on "cgroup" psuedo filesystem.

Too much indentation.

s/cpu/CPU/g please.

> +
> endchoice

>
> +config RESOURCE_COUNTERS
> + bool "Resource counters"
> + help
> +   This option enables controller independent resource accounting
> +   infrastructure that works with cgroups

Use tab + 2 spaces to indent help text.

> + depends on CGROUPS
> +
> config SYSFS_DEPRECATED
>   bool "Create deprecated sysfs files"
>   default y

~Randy
Phaedrus says that Quality is about caring.

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

---

Subject: Re: [PATCH 3/5] Cleanup code under CONFIG_FAIR_GROUP_SCHED
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 17:07:11 GMT
View Forum Message <> Reply to Message

On Mon, Sep 24, 2007 at 09:53:44AM -0700, Randy Dunlap wrote:
> > config FAIR_GROUP_SCHED
> > - bool "Fair group scheduler"
> > - depends on EXPERIMENTAL && CONTAINERS
> > + bool "Fair group cpu scheduler"
>
> Can we have "CPU" instead of "cpu" ?

Sounds good. Will add to my follow-on.patch (and will send out after
others have had a chance to comment).

--
Regards,
vatsa

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

---

Subject: Re: [PATCH 4/5] Add fair-user scheduler
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 17:09:49 GMT
View Forum Message <> Reply to Message

On Mon, Sep 24, 2007 at 09:56:41AM -0700, Randy Dunlap wrote:
> > +choice
> > + depends on FAIR_GROUP_SCHED
> > + prompt "Basis for grouping tasks"
> > + default FAIR_USER_SCHED
> > +
> > + config FAIR_USER_SCHED
> > +   bool "user id"
> > +   help
> > +   This option will choose userid as the basis for grouping
> > +   tasks, thus providing equal cpu bandwidth to each user.
>
> s/cpu/CPU/g please.

ok, sure.

> and the "bool", "help" and help text are indented by one tab too much.

will take care.

--
Regards,
vatsa

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

Subject: Re: [PATCH 5/5] Add fair "control groups" scheduler
Posted by Srivatsa Vaddagiri on Mon, 24 Sep 2007 17:10:53 GMT
View Forum Message <> Reply to Message

On Mon, Sep 24, 2007 at 09:58:15AM -0700, Randy Dunlap wrote:
> > + config FAIR_CGROUP_SCHED
> > + bool "Control groups"
> > + depends on CGROUPS
> > + help
> > +   This option allows you to create arbitrary task groups
> > +   using the "cgroup" psuedo filesystem and control
> > +   the cpu bandwidth allocated to each such task group.
> > +   Refer to Documentation/cgroups.txt for more information
> > +   on "cgroup" psuedo filesystem.
>

> Too much indentation.
>
> s/cpu/CPU/g please.

will fix both.

> > +config RESOURCE_COUNTERS
> > + bool "Resource counters"
> > + help
> > + This option enables controller independent resource accounting
> > + infrastructure that works with cgroups
>
> Use tab + 2 spaces to indent help text.

This one was there before in -mm ..I just moved it below in the Kconfig file. Neverthless will fix the coding-style here as well.

Thanks for your reviews!

--
Regards,
vatsa

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

---

Subject: Re: [PATCH 4/5] Add fair-user scheduler
Posted by Ingo Molnar on Mon, 24 Sep 2007 18:01:55 GMT
View Forum Message <> Reply to Message

* Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com> wrote:

> Enable user-id based fair group scheduling. This is usefull for anyone
> who wants to test the group scheduler w/o having to enable
> CONFIG_CGROUPS.

excellent! I'll make this default-enabled.

Ingo

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

---

Subject: Re: [PATCH 4/5] Add fair-user scheduler
Posted by roel on Mon, 24 Sep 2007 23:39:39 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> Enable user-id based fair group scheduling. This is usefull for anyone
> who wants to test the group scheduler w/o having to enable
> CONFIG_CGROUPS.
>
> A separate scheduling group (i.e struct task_grp) is automatically created for
> every new user added to the system. Upon uid change for a task, it is made to
> move to the corresponding scheduling group.
>
> A /proc tunable (/proc/root_user_share) is also provided to tune root
> user's quota of cpu bandwidth.
>
> Signed-off-by : Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com>
> Signed-off-by : Dhaval Giani <dhaval@linux.vnet.ibm.com>
>
>
> ---
> include/linux/sched.h | 4 +++
> init/Kconfig | 13 ++++++++++
> kernel/sched.c | 9 ++++++++
> kernel/sched_debug.c | 52 ++
> kernel/user.c | 43 ++
> 5 files changed, 121 insertions(+)
>
> Index: linux-2.6.23-rc6/include/linux/sched.h
> ===
> --- linux-2.6.23-rc6.orig/include/linux/sched.h
> +++ linux-2.6.23-rc6/include/linux/sched.h
> @@ -596,6 +596,10 @@ struct user_struct {
>   /* Hash table maintenance information */
>   struct hlist_node uidhash_node;
>   uid_t uid;
> +
> +#ifdef CONFIG_FAIR_USER_SCHED
> + struct task_grp *tg;
> +#endif
> };
>
> extern struct user_struct *find_user(uid_t);
> Index: linux-2.6.23-rc6/init/Kconfig
> ===
> --- linux-2.6.23-rc6.orig/init/Kconfig
> +++ linux-2.6.23-rc6/init/Kconfig
> @@ -289,6 +289,19 @@ config FAIR_GROUP_SCHED
>   This feature lets cpu scheduler recognize task groups and control cpu

```
>     bandwidth allocation to such task groups.
>
> +choice
> + depends on FAIR_GROUP_SCHED
> + prompt "Basis for grouping tasks"
> + default FAIR_USER_SCHED
> +
> +  config FAIR_USER_SCHED
> +   bool "user id"
> +   help
> +     This option will choose userid as the basis for grouping
> +    tasks, thus providing equal cpu bandwidth to each user.
> +
> +endchoice
> +
>  config SYSFS_DEPRECATED
>   bool "Create deprecated sysfs files"
>   default y
> Index: linux-2.6.23-rc6/kernel/sched.c
> ===================================================================
> --- linux-2.6.23-rc6.orig/kernel/sched.c
> +++ linux-2.6.23-rc6/kernel/sched.c
> @@ -199,7 +199,12 @@ struct task_grp init_task_grp =  {
>       .cfs_rq = init_cfs_rq_p,
>        };
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> +#define INIT_TASK_GRP_LOAD 2*NICE_0_LOAD
> +#else
>  #define INIT_TASK_GRP_LOAD NICE_0_LOAD
> +#endif
> +
>  static int init_task_grp_load = INIT_TASK_GRP_LOAD;
>
>  /* return group to which a task belongs */
> @@ -207,7 +212,11 @@ static inline struct task_grp *task_grp(
>  {
>   struct task_grp *tg;
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> + tg = p->user->tg;
> +#else
>   tg  = &init_task_grp;
> +#endif
>
>   return tg;
>  }
> Index: linux-2.6.23-rc6/kernel/sched_debug.c
```

```
> ===================================================================
> --- linux-2.6.23-rc6.orig/kernel/sched_debug.c
> +++ linux-2.6.23-rc6/kernel/sched_debug.c
> @@ -214,6 +214,49 @@ static void sysrq_sched_debug_show(void)
>   sched_debug_show(NULL, NULL);
>  }
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> +
> +static DEFINE_MUTEX(root_user_share_mutex);
> +
> +static int
> +root_user_share_read_proc(char *page, char **start, off_t off, int count,
> +     int *eof, void *data)
> +{
> + int len;
> +
> + len = sprintf(page, "%d\n", init_task_grp_load);
> +
> + return len;
> +}
```

or use this oneliner:

```
 return sprintf(page, "%d\n", init_task_grp_load);
```

```
> +
> +static int
> +root_user_share_write_proc(struct file *file, const char __user *buffer,
> +     unsigned long count, void *data)
> +{
> + unsigned long shares;
> + char kbuf[sizeof(unsigned long)+1];
> + int rc = 0;
> +
> + if (copy_from_user(kbuf, buffer, sizeof(kbuf)))
> +  return -EFAULT;
> +
> + shares = simple_strtoul(kbuf, NULL, 0);
> +
> + if (!shares)
> +  shares = NICE_0_LOAD;
> +
> + mutex_lock(&root_user_share_mutex);
> +
> + init_task_grp_load = shares;
> + rc = sched_group_set_shares(&init_task_grp, shares);
> +
```

```
> + mutex_unlock(&root_user_share_mutex);
> +
> + return (rc < 0 ? rc : count);
> +}
> +
> +#endif /* CONFIG_FAIR_USER_SCHED */
> +
>  static int sched_debug_open(struct inode *inode, struct file *filp)
>  {
>   return single_open(filp, sched_debug_show, NULL);
> @@ -236,6 +279,15 @@ static int __init init_sched_debug_procf
>
>   pe->proc_fops = &sched_debug_fops;
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> + pe = create_proc_entry("root_user_share", 0644, NULL);
> + if (!pe)
> +  return -ENOMEM;
> +
> + pe->read_proc = root_user_share_read_proc;
> + pe->write_proc = root_user_share_write_proc;
> +#endif
> +
>   return 0;
>  }
>
> Index: linux-2.6.23-rc6/kernel/user.c
> ===================================================================
> --- linux-2.6.23-rc6.orig/kernel/user.c
> +++ linux-2.6.23-rc6/kernel/user.c
> @@ -50,8 +50,41 @@ struct user_struct root_user = {
>   .uid_keyring = &root_user_keyring,
>   .session_keyring = &root_session_keyring,
>  #endif
> +#ifdef CONFIG_FAIR_USER_SCHED
> + .tg  = &init_task_grp,
> +#endif
>  };
>
> +#ifdef CONFIG_FAIR_USER_SCHED
> +static void sched_destroy_user(struct user_struct *up)
> +{
> + sched_destroy_group(up->tg);
> +}
> +
> +static int sched_create_user(struct user_struct *up)
> +{
> + int rc = 0;
```

```
> +
> + up->tg = sched_create_group();
> + if (IS_ERR(up->tg))
> +  rc = -ENOMEM;
> +
> + return rc;
> +}
> +
> +static void sched_switch_user(struct task_struct *p)
> +{
> + sched_move_task(p);
> +}
> +
> +#else /* CONFIG_FAIR_USER_SCHED */
> +
> +static void sched_destroy_user(struct user_struct *up) { }
> +static int sched_create_user(struct user_struct *up) { return 0; }
> +static void sched_switch_user(struct task_struct *p) { }
> +
> +#endif /* CONFIG_FAIR_USER_SCHED */
> +
>  /*
>   * These routines must be called with the uidhash spinlock held!
>   */
> @@ -109,6 +142,7 @@ void free_uid(struct user_struct *up)
>   if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
>    uid_hash_remove(up);
>    spin_unlock_irqrestore(&uidhash_lock, flags);
> +  sched_destroy_user(up);
>    key_put(up->uid_keyring);
>    key_put(up->session_keyring);
>    kmem_cache_free(uid_cachep, up);
> @@ -150,6 +184,13 @@ struct user_struct * alloc_uid(struct us
>     return NULL;
>    }
>
> +  if (sched_create_user(new) < 0) {
> +   key_put(new->uid_keyring);
> +   key_put(new->session_keyring);
> +   kmem_cache_free(uid_cachep, new);
> +   return NULL;
> +  }
> +
>    /*
>     * Before adding this, check whether we raced
>     * on adding the same user already..
> @@ -157,6 +198,7 @@ struct user_struct * alloc_uid(struct us
>    spin_lock_irq(&uidhash_lock);
```

```
>    up = uid_hash_find(uid, hashent);
>    if (up) {
> +   sched_destroy_user(new);
>     key_put(new->uid_keyring);
>     key_put(new->session_keyring);
>     kmem_cache_free(uid_cachep, new);
> @@ -184,6 +226,7 @@ void switch_uid(struct user_struct *new_
>   atomic_dec(&old_user->processes);
>   switch_uid_keyring(new_user);
>   current->user = new_user;
> + sched_switch_user(current);
>
>   /*
>    * We need to synchronize with __sigqueue_alloc()
>
```

Subject: Re: [PATCH 4/5] Add fair-user scheduler
Posted by Srivatsa Vaddagiri on Tue, 25 Sep 2007 02:02:28 GMT
View Forum Message <> Reply to Message

On Tue, Sep 25, 2007 at 01:39:39AM +0200, roel wrote:
```
> > +static int
> > +root_user_share_read_proc(char *page, char **start, off_t off, int count,
> > +     int *eof, void *data)
> > +{
> > + int len;
> > +
> > + len = sprintf(page, "%d\n", init_task_grp_load);
> > +
> > + return len;
> > +}
>
> or use this oneliner:
>
>  return sprintf(page, "%d\n", init_task_grp_load);
```

Looks good. Will fix this in a follow-on.patch.

Thanks!

--
Regards,

vatsa

_____

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