Subject: Re: [RFC][PATCH 0/6] Add group fairness to CFS - v1
Posted by Ingo Molnar on Mon, 11 Jun 2007 19:37:35 GMT
View Forum Message <> Reply to Message

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

> Ingo,
>  Here's an update of the group fairness patch I have been
> working on. Its against CFS v16 (sched-cfs-v2.6.22-rc4-mm2-v16.patch).

thanks!

> The core idea is to reuse much of CFS logic to apply fairness at
> higher hierarchical levels (user, container etc). In this regard CFS
> engine has been modified to deal with generic 'schedulable entities'.
> The patches introduce two essential structures in CFS core:
>
>  - struct sched_entity
>   - represents a schedulable entity in a hierarchy. Task
>     is the lowest element in this hierarchy. Its ancestors
>     could be user, container etc. This structure stores
>     essential attributes/execution-history (wait_runtime etc)
>     which is required by CFS engine to provide fairness between
>     'struct sched_entities' at the same hierarchy.
>
>  - struct lrq
>   - represents (per-cpu) runqueue in which ready-to-run
>     'struct sched_entities' are queued. The fair clock
>     calculation is split to be per 'struct lrq'.
>
> Here's a brief description of the patches to follow:
>
> Patches 1-3 introduce the essential changes in CFS core to support
> this concept. They rework existing code w/o any (intended!) change in
> functionality.

i currently have these 3 patches applied to the CFS queue and it's
looking pretty good so far! If it continues to be problem-free i'll
release them as part of -v17, just to check that they truly have no bad
side-effects (they shouldnt). Then #4 can go into -v18.

i've attached my current -v17 tree - it should apply mostly cleanly
ontop of the -mm queue (with a minor number of fixups). Could you
refactor the remaining 3 patches ontop of this base? There's some
rejects in the last 3 patches due to the update_load_fair() change.

> Patch 4 fixes some bad interaction between SCHED_RT and SCHED_NORMAL
> tasks in current CFS.

btw., the plan here is to turn off 'bit 0' in sched_features: i.e. to
use the precise statistics to calculate lrq->cpu_load[], not the
timer-irq-sampled imprecise statistics. Dmitry has fixed a couple of
bugs in it that made it not work too well in previous CFS versions, but
now we are ready to turn it on for -v17. (indeed in my tree it's already
turned on - i.e. sched_features defaults to '14')

> Patch 5 introduces basic changes in CFS core to support group
> fairness.
>
> Patch 6 hooks up scheduler with container patches in mm (as an
> interface for task-grouping functionality).

ok. Kirill, how do you like Srivatsa's current approach? Would be nice
to kill two birds with the same stone, if possible :-)

> Note: I have noticed that running lat_ctx in a loop for 10 times
> doesnt give me good results. Basically I expected the loop to take
> same time for both users (when run simultaneously), whereas it was
> taking different times for different users. I think this can be solved
> by increasing sysctl_sched_runtime_limit at group level (to remeber
> execution history over a longer period).

you'll get the best hackbench results by using SCHED_BATCH:

   chrt -b 0 ./hackbench 10

or indeed increasing the runtime_limit would work too.

 Ingo

Index: linux/Makefile
===================================================================
--- linux.orig/Makefile
+++ linux/Makefile
@@ -1,7 +1,7 @@
 VERSION = 2
 PATCHLEVEL = 6
 SUBLEVEL = 21
-EXTRAVERSION = .4-cfs-v16
+EXTRAVERSION = .4-cfs-v17
 NAME = Nocturnal Monster Puppy

 # *DOCUMENTATION*
Index: linux/fs/proc/array.c
===================================================================
--- linux.orig/fs/proc/array.c

```
+++ linux/fs/proc/array.c
@@ -319,7 +319,7 @@ static clock_t task_utime(struct task_st
   * Use CFS's precise accounting, if available:
   */
  if (!(sysctl_sched_features & 128)) {
-  u64 temp = (u64)nsec_to_clock_t(p->sum_exec_runtime);
+  u64 temp = (u64)nsec_to_clock_t(p->se.sum_exec_runtime);

   if (total) {
    temp *= utime;
@@ -341,7 +341,7 @@ static clock_t task_stime(struct task_st
   * by userspace grows monotonically - apps rely on that):
   */
  if (!(sysctl_sched_features & 128))
-  stime = nsec_to_clock_t(p->sum_exec_runtime) - task_utime(p);
+  stime = nsec_to_clock_t(p->se.sum_exec_runtime) - task_utime(p);

  return stime;
 }
Index: linux/include/linux/sched.h
===================================================================
--- linux.orig/include/linux/sched.h
+++ linux/include/linux/sched.h
@@ -534,8 +534,7 @@ struct signal_struct {

 #define rt_prio(prio)  unlikely((prio) < MAX_RT_PRIO)
 #define rt_task(p)  rt_prio((p)->prio)
-#define batch_task(p)  (unlikely((p)->policy == SCHED_BATCH))
-#define is_rt_policy(p)  ((p) != SCHED_NORMAL && (p) != SCHED_BATCH)
+#define is_rt_policy(p)  ((p) == SCHED_FIFO || (p) == SCHED_RR)
 #define has_rt_policy(p) unlikely(is_rt_policy((p)->policy))

 /*
@@ -819,6 +818,29 @@ struct sched_class {
  void (*task_new) (struct rq *rq, struct task_struct *p);
 };

+/* CFS stats for a schedulable entity (task, task-group etc) */
+struct sched_entity {
+ int load_weight; /* for niceness load balancing purposes */
+ int on_rq;
+ struct rb_node run_node;
+ u64 wait_start_fair;
+ u64 wait_start;
+ u64 exec_start;
+ u64 sleep_start, sleep_start_fair;
+ u64 block_start;
+ u64 sleep_max;
```

```
+ u64 block_max;
+ u64 exec_max;
+ u64 wait_max;
+ u64 last_ran;
+
+ s64 wait_runtime;
+ u64 sum_exec_runtime;
+ s64 fair_key;
+ s64 sum_wait_runtime, sum_sleep_runtime;
+ unsigned long wait_runtime_overruns, wait_runtime_underruns;
+};
+
 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
  struct thread_info *thread_info;
@@ -833,33 +855,15 @@ struct task_struct {
  int oncpu;
 #endif
 #endif
- int load_weight; /* for niceness load balancing purposes */

  int prio, static_prio, normal_prio;
- int on_rq;
  struct list_head run_list;
- struct rb_node run_node;
+ struct sched_entity se;

  unsigned short ioprio;
 #ifdef CONFIG_BLK_DEV_IO_TRACE
  unsigned int btrace_seq;
 #endif
- /* CFS scheduling class statistics fields: */
- u64 wait_start_fair;
- u64 wait_start;
- u64 exec_start;
- u64 sleep_start, sleep_start_fair;
- u64 block_start;
- u64 sleep_max;
- u64 block_max;
- u64 exec_max;
- u64 wait_max;
-
- s64 wait_runtime;
- u64 sum_exec_runtime;
- s64 fair_key;
- s64 sum_wait_runtime, sum_sleep_runtime;
- unsigned long wait_runtime_overruns, wait_runtime_underruns;
```

```
  unsigned long policy;
  cpumask_t cpus_allowed;
Index: linux/kernel/exit.c
===================================================================
--- linux.orig/kernel/exit.c
+++ linux/kernel/exit.c
@@ -112,7 +112,7 @@ static void __exit_signal(struct task_st
   sig->maj_flt += tsk->maj_flt;
   sig->nvcsw += tsk->nvcsw;
   sig->nivcsw += tsk->nivcsw;
-  sig->sum_sched_runtime += tsk->sum_exec_runtime;
+  sig->sum_sched_runtime += tsk->se.sum_exec_runtime;
   sig = NULL; /* Marker for below. */
  }

Index: linux/kernel/posix-cpu-timers.c
===================================================================
--- linux.orig/kernel/posix-cpu-timers.c
+++ linux/kernel/posix-cpu-timers.c
@@ -249,7 +249,7 @@ static int cpu_clock_sample_group_locked
   cpu->sched = p->signal->sum_sched_runtime;
   /* Add in each other live thread.  */
   while ((t = next_thread(t)) != p) {
-   cpu->sched += t->sum_exec_runtime;
+   cpu->sched += t->se.sum_exec_runtime;
   }
   cpu->sched += sched_ns(p);
   break;
@@ -467,7 +467,7 @@ static void cleanup_timers(struct list_h
 void posix_cpu_timers_exit(struct task_struct *tsk)
 {
  cleanup_timers(tsk->cpu_timers,
-        tsk->utime, tsk->stime, tsk->sum_exec_runtime);
+        tsk->utime, tsk->stime, tsk->se.sum_exec_runtime);

 }
 void posix_cpu_timers_exit_group(struct task_struct *tsk)
@@ -475,7 +475,7 @@ void posix_cpu_timers_exit_group(struct
  cleanup_timers(tsk->signal->cpu_timers,
         cputime_add(tsk->utime, tsk->signal->utime),
         cputime_add(tsk->stime, tsk->signal->stime),
-        tsk->sum_exec_runtime + tsk->signal->sum_sched_runtime);
+       tsk->se.sum_exec_runtime + tsk->signal->sum_sched_runtime);
 }


@@ -536,7 +536,7 @@ static void process_timer_rebalance(stru
   nsleft = max_t(unsigned long long, nsleft, 1);
```

```
   do {
    if (likely(!(t->flags & PF_EXITING))) {
-    ns = t->sum_exec_runtime + nsleft;
+    ns = t->se.sum_exec_runtime + nsleft;
     if (t->it_sched_expires == 0 ||
         t->it_sched_expires > ns) {
      t->it_sched_expires = ns;
@@ -1004,7 +1004,7 @@ static void check_thread_timers(struct t
   struct cpu_timer_list *t = list_entry(timers->next,
             struct cpu_timer_list,
             entry);
-  if (!--maxfire || tsk->sum_exec_runtime < t->expires.sched) {
+  if (!--maxfire || tsk->se.sum_exec_runtime < t->expires.sched) {
    tsk->it_sched_expires = t->expires.sched;
    break;
   }
@@ -1049,7 +1049,7 @@ static void check_process_timers(struct
  do {
   utime = cputime_add(utime, t->utime);
   stime = cputime_add(stime, t->stime);
-  sum_sched_runtime += t->sum_exec_runtime;
+  sum_sched_runtime += t->se.sum_exec_runtime;
   t = next_thread(t);
  } while (t != tsk);
  ptime = cputime_add(utime, stime);
@@ -1208,7 +1208,7 @@ static void check_process_timers(struct
     t->it_virt_expires = ticks;
    }

-   sched = t->sum_exec_runtime + sched_left;
+   sched = t->se.sum_exec_runtime + sched_left;
    if (sched_expires && (t->it_sched_expires == 0 ||
          t->it_sched_expires > sched)) {
     t->it_sched_expires = sched;
@@ -1300,7 +1300,7 @@ void run_posix_cpu_timers(struct task_st

  if (UNEXPIRED(prof) && UNEXPIRED(virt) &&
      (tsk->it_sched_expires == 0 ||
-      tsk->sum_exec_runtime < tsk->it_sched_expires))
+      tsk->se.sum_exec_runtime < tsk->it_sched_expires))
   return;

 #undef UNEXPIRED
Index: linux/kernel/sched.c
===================================================================
--- linux.orig/kernel/sched.c
+++ linux/kernel/sched.c
@@ -113,6 +113,23 @@ struct prio_array {
```

```
  struct list_head queue[MAX_RT_PRIO];
 };

+/* CFS-related fields in a runqueue */
+struct lrq {
+ unsigned long raw_weighted_load;
+ #define CPU_LOAD_IDX_MAX 5
+ unsigned long cpu_load[CPU_LOAD_IDX_MAX];
+ unsigned long nr_load_updates;
+
+ u64 fair_clock, delta_fair_clock;
+ u64 exec_clock, delta_exec_clock;
+ s64 wait_runtime;
+ unsigned long wait_runtime_overruns, wait_runtime_underruns;
+
+ struct rb_root tasks_timeline;
+ struct rb_node *rb_leftmost;
+ struct rb_node *rb_load_balance_curr;
+};
+
 /*
  * This is the main, per-CPU runqueue data structure.
  *
@@ -128,12 +145,9 @@ struct rq {
   * remote CPUs use both these fields when doing load calculation.
   */
  long nr_running;
- unsigned long raw_weighted_load;
- #define CPU_LOAD_IDX_MAX 5
- unsigned long cpu_load[CPU_LOAD_IDX_MAX];
+ struct lrq lrq;

  u64 nr_switches;
- unsigned long nr_load_updates;

  /*
   * This is part of a global counter where only the total sum
@@ -149,10 +163,6 @@ struct rq {

  u64 clock, prev_clock_raw;
  s64 clock_max_delta;
- u64 fair_clock, delta_fair_clock;
- u64 exec_clock, delta_exec_clock;
- s64 wait_runtime;
- unsigned long wait_runtime_overruns, wait_runtime_underruns;

  unsigned int clock_warps, clock_overflows;
  unsigned int clock_unstable_events;
```

```
@@ -163,10 +173,6 @@ struct rq {
  int rt_load_balance_idx;
  struct list_head *rt_load_balance_head, *rt_load_balance_curr;

- struct rb_root tasks_timeline;
- struct rb_node *rb_leftmost;
- struct rb_node *rb_load_balance_curr;
-
  atomic_t nr_iowait;

 #ifdef CONFIG_SMP
@@ -543,13 +549,13 @@ const int prio_to_weight[40] = {
 static inline void
 inc_raw_weighted_load(struct rq *rq, const struct task_struct *p)
 {
- rq->raw_weighted_load += p->load_weight;
+ rq->lrq.raw_weighted_load += p->se.load_weight;
 }

 static inline void
 dec_raw_weighted_load(struct rq *rq, const struct task_struct *p)
 {
- rq->raw_weighted_load -= p->load_weight;
+ rq->lrq.raw_weighted_load -= p->se.load_weight;
 }

 static inline void inc_nr_running(struct task_struct *p, struct rq *rq)
@@ -575,22 +581,22 @@ static void activate_task(struct rq *rq,

 static void set_load_weight(struct task_struct *p)
 {
- task_rq(p)->wait_runtime -= p->wait_runtime;
- p->wait_runtime = 0;
+ task_rq(p)->lrq.wait_runtime -= p->se.wait_runtime;
+ p->se.wait_runtime = 0;

  if (has_rt_policy(p)) {
-  p->load_weight = prio_to_weight[0] * 2;
+  p->se.load_weight = prio_to_weight[0] * 2;
   return;
  }
  /*
   * SCHED_IDLEPRIO tasks get minimal weight:
   */
  if (p->policy == SCHED_IDLEPRIO) {
-  p->load_weight = 1;
+  p->se.load_weight = 1;
   return;
```

```
  }

- p->load_weight = prio_to_weight[p->static_prio - MAX_RT_PRIO];
+ p->se.load_weight = prio_to_weight[p->static_prio - MAX_RT_PRIO];
 }

 static void enqueue_task(struct rq *rq, struct task_struct *p, int wakeup)
@@ -599,7 +605,7 @@ static void enqueue_task(struct rq *rq,

  sched_info_queued(p);
  p->sched_class->enqueue_task(rq, p, wakeup, now);
- p->on_rq = 1;
+ p->se.on_rq = 1;
 }

 static void dequeue_task(struct rq *rq, struct task_struct *p, int sleep)
@@ -607,7 +613,7 @@ static void dequeue_task(struct rq *rq,
  u64 now = rq_clock(rq);

  p->sched_class->dequeue_task(rq, p, sleep, now);
- p->on_rq = 0;
+ p->se.on_rq = 0;
 }

 /*
@@ -695,7 +701,7 @@ inline int task_curr(const struct task_s
 /* Used instead of source_load when we know the type == 0 */
 unsigned long weighted_cpuload(const int cpu)
 {
- return cpu_rq(cpu)->raw_weighted_load;
+ return cpu_rq(cpu)->lrq.raw_weighted_load;
 }

 #ifdef CONFIG_SMP
@@ -712,18 +718,18 @@ void set_task_cpu(struct task_struct *p,
  u64 clock_offset, fair_clock_offset;

  clock_offset = old_rq->clock - new_rq->clock;
- fair_clock_offset = old_rq->fair_clock - new_rq->fair_clock;
+ fair_clock_offset = old_rq->lrq.fair_clock - new_rq->lrq.fair_clock;

- if (p->wait_start)
-  p->wait_start -= clock_offset;
- if (p->wait_start_fair)
-  p->wait_start_fair -= fair_clock_offset;
- if (p->sleep_start)
-  p->sleep_start -= clock_offset;
- if (p->block_start)
```

```
-  p->block_start -= clock_offset;
- if (p->sleep_start_fair)
-  p->sleep_start_fair -= fair_clock_offset;
+ if (p->se.wait_start)
+  p->se.wait_start -= clock_offset;
+ if (p->se.wait_start_fair)
+  p->se.wait_start_fair -= fair_clock_offset;
+ if (p->se.sleep_start)
+  p->se.sleep_start -= clock_offset;
+ if (p->se.block_start)
+  p->se.block_start -= clock_offset;
+ if (p->se.sleep_start_fair)
+  p->se.sleep_start_fair -= fair_clock_offset;

  task_thread_info(p)->cpu = new_cpu;

@@ -751,7 +757,7 @@ migrate_task(struct task_struct *p, int
   * If the task is not on a runqueue (and not running), then
   * it is sufficient to simply update the task's cpu field.
   */
- if (!p->on_rq && !task_running(rq, p)) {
+ if (!p->se.on_rq && !task_running(rq, p)) {
   set_task_cpu(p, dest_cpu);
   return 0;
  }
@@ -782,7 +788,7 @@ void wait_task_inactive(struct task_stru
 repeat:
  rq = task_rq_lock(p, &flags);
  /* Must be off runqueue entirely, not preempted. */
- if (unlikely(p->on_rq || task_running(rq, p))) {
+ if (unlikely(p->se.on_rq || task_running(rq, p))) {
   /* If it's preempted, we yield.  It could be a while. */
   preempted = !task_running(rq, p);
   task_rq_unlock(rq, &flags);
@@ -830,9 +836,9 @@ static inline unsigned long source_load(
  struct rq *rq = cpu_rq(cpu);

  if (type == 0)
-  return rq->raw_weighted_load;
+  return rq->lrq.raw_weighted_load;

- return min(rq->cpu_load[type-1], rq->raw_weighted_load);
+ return min(rq->lrq.cpu_load[type-1], rq->lrq.raw_weighted_load);
 }

 /*
@@ -844,9 +850,9 @@ static inline unsigned long target_load(
  struct rq *rq = cpu_rq(cpu);
```

```
  if (type == 0)
-  return rq->raw_weighted_load;
+  return rq->lrq.raw_weighted_load;

- return max(rq->cpu_load[type-1], rq->raw_weighted_load);
+ return max(rq->lrq.cpu_load[type-1], rq->lrq.raw_weighted_load);
 }

 /*
@@ -857,7 +863,7 @@ static inline unsigned long cpu_avg_load
  struct rq *rq = cpu_rq(cpu);
  unsigned long n = rq->nr_running;

- return n ? rq->raw_weighted_load / n : SCHED_LOAD_SCALE;
+ return n ? rq->lrq.raw_weighted_load / n : SCHED_LOAD_SCALE;
 }

 /*
@@ -1078,7 +1084,7 @@ static int try_to_wake_up(struct task_st
  if (!(old_state & state))
   goto out;

- if (p->on_rq)
+ if (p->se.on_rq)
   goto out_running;

  cpu = task_cpu(p);
@@ -1133,11 +1139,11 @@ static int try_to_wake_up(struct task_st
     * of the current CPU:
     */
    if (sync)
-    tl -= current->load_weight;
+    tl -= current->se.load_weight;

    if ((tl <= load &&
     tl + target_load(cpu, idx) <= tl_per_task) ||
-    100*(tl + p->load_weight) <= imbalance*load) {
+         100*(tl + p->se.load_weight) <= imbalance*load) {
     /*
      * This domain has SD_WAKE_AFFINE and
      * p is cache cold in this domain, and
@@ -1171,7 +1177,7 @@ out_set_cpu:
   old_state = p->state;
   if (!(old_state & state))
    goto out;
-  if (p->on_rq)
+  if (p->se.on_rq)
```

```
    goto out_running;

   this_cpu = smp_processor_id();
@@ -1235,18 +1241,18 @@ static void task_running_tick(struct rq
  */
 static void __sched_fork(struct task_struct *p)
 {
- p->wait_start_fair = p->wait_start = p->exec_start = 0;
- p->sum_exec_runtime = 0;
+ p->se.wait_start_fair = p->se.wait_start = p->se.exec_start = 0;
+ p->se.sum_exec_runtime = 0;

- p->wait_runtime = 0;
+ p->se.wait_runtime = 0;

- p->sum_wait_runtime = p->sum_sleep_runtime = 0;
- p->sleep_start = p->sleep_start_fair = p->block_start = 0;
- p->sleep_max = p->block_max = p->exec_max = p->wait_max = 0;
- p->wait_runtime_overruns = p->wait_runtime_underruns = 0;
+ p->se.sum_wait_runtime = p->se.sum_sleep_runtime = 0;
+ p->se.sleep_start = p->se.sleep_start_fair = p->se.block_start = 0;
+ p->se.sleep_max = p->se.block_max = p->se.exec_max = p->se.wait_max = 0;
+ p->se.wait_runtime_overruns = p->se.wait_runtime_underruns = 0;

  INIT_LIST_HEAD(&p->run_list);
- p->on_rq = 0;
+ p->se.on_rq = 0;
  p->nr_switches = 0;

  /*
@@ -1317,7 +1323,7 @@ void fastcall wake_up_new_task(struct ta
  p->prio = effective_prio(p);

  if (!sysctl_sched_child_runs_first || (clone_flags & CLONE_VM) ||
-   task_cpu(p) != this_cpu || !current->on_rq) {
+   task_cpu(p) != this_cpu || !current->se.on_rq) {
   activate_task(rq, p, 0);
  } else {
   /*
@@ -1332,7 +1338,7 @@ void fastcall wake_up_new_task(struct ta

 void sched_dead(struct task_struct *p)
 {
- WARN_ON_ONCE(p->on_rq);
+ WARN_ON_ONCE(p->se.on_rq);
 }

 /**
```

```
@@ -1542,17 +1548,17 @@ static void update_load_fair(struct rq *
  u64 fair_delta64, exec_delta64, tmp64;
  unsigned int i, scale;

- this_rq->nr_load_updates++;
- if (!(sysctl_sched_features & 64)) {
-  this_load = this_rq->raw_weighted_load;
+ this_rq->lrq.nr_load_updates++;
+ if (sysctl_sched_features & 64) {
+  this_load = this_rq->lrq.raw_weighted_load;
   goto do_avg;
  }

- fair_delta64 = this_rq->delta_fair_clock + 1;
- this_rq->delta_fair_clock = 0;
+ fair_delta64 = this_rq->lrq.delta_fair_clock + 1;
+ this_rq->lrq.delta_fair_clock = 0;

- exec_delta64 = this_rq->delta_exec_clock + 1;
- this_rq->delta_exec_clock = 0;
+ exec_delta64 = this_rq->lrq.delta_exec_clock + 1;
+ this_rq->lrq.delta_exec_clock = 0;

  if (fair_delta64 > (u64)LONG_MAX)
   fair_delta64 = (u64)LONG_MAX;
@@ -1577,10 +1583,10 @@ do_avg:

   /* scale is effectively 1 << i now, and >> i divides by scale */

-  old_load = this_rq->cpu_load[i];
+  old_load = this_rq->lrq.cpu_load[i];
   new_load = this_load;

-  this_rq->cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
+  this_rq->lrq.cpu_load[i] = (old_load*(scale-1) + new_load) >> i;
  }
 }

@@ -1836,7 +1842,8 @@ next:
   * skip a task if it will be the highest priority task (i.e. smallest
   * prio value) on its new queue regardless of its load weight
   */
- skip_for_load = (p->load_weight >> 1) > rem_load_move + SCHED_LOAD_SCALE_FUZZ;
+ skip_for_load = (p->se.load_weight >> 1) > rem_load_move +
+        SCHED_LOAD_SCALE_FUZZ;
  if (skip_for_load && p->prio < this_best_prio)
   skip_for_load = !best_prio_seen && p->prio == best_prio;
  if (skip_for_load ||
```

```
@@ -1849,7 +1856,7 @@ next:

  pull_task(busiest, p, this_rq, this_cpu);
  pulled++;
- rem_load_move -= p->load_weight;
+ rem_load_move -= p->se.load_weight;

  /*
   * We only want to steal up to the prescribed number of tasks
@@ -1946,7 +1953,7 @@ find_busiest_group(struct sched_domain *

    avg_load += load;
    sum_nr_running += rq->nr_running;
-   sum_weighted_load += rq->raw_weighted_load;
+   sum_weighted_load += rq->lrq.raw_weighted_load;
   }

   /*
@@ -2178,11 +2185,12 @@ find_busiest_queue(struct sched_group *g

   rq = cpu_rq(i);

-  if (rq->nr_running == 1 && rq->raw_weighted_load > imbalance)
+  if (rq->nr_running == 1 &&
+      rq->lrq.raw_weighted_load > imbalance)
    continue;

-  if (rq->raw_weighted_load > max_load) {
-   max_load = rq->raw_weighted_load;
+  if (rq->lrq.raw_weighted_load > max_load) {
+   max_load = rq->lrq.raw_weighted_load;
    busiest = rq;
   }
  }
@@ -2607,9 +2615,9 @@ unsigned long long task_sched_runtime(st
  struct rq *rq;

  rq = task_rq_lock(p, &flags);
- ns = p->sum_exec_runtime;
+ ns = p->se.sum_exec_runtime;
  if (rq->curr == p) {
-  delta_exec = rq_clock(rq) - p->exec_start;
+  delta_exec = rq_clock(rq) - p->se.exec_start;
   if ((s64)delta_exec > 0)
    ns += delta_exec;
  }
@@ -3299,7 +3307,7 @@ void rt_mutex_setprio(struct task_struct
  rq = task_rq_lock(p, &flags);
```

```
  oldprio = p->prio;
- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
  if (on_rq)
   dequeue_task(rq, p, 0);

@@ -3352,7 +3360,7 @@ void set_user_nice(struct task_struct *p
   p->static_prio = NICE_TO_PRIO(nice);
   goto out_unlock;
  }
- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
  if (on_rq) {
   dequeue_task(rq, p, 0);
   dec_raw_weighted_load(rq, p);
@@ -3489,12 +3497,13 @@ static inline struct task_struct *find_p
 static void
 __setscheduler(struct rq *rq, struct task_struct *p, int policy, int prio)
 {
- BUG_ON(p->on_rq);
+ BUG_ON(p->se.on_rq);

  p->policy = policy;
  switch (p->policy) {
  case SCHED_NORMAL:
  case SCHED_BATCH:
+ case SCHED_ISO:
  case SCHED_IDLEPRIO:
   p->sched_class = &fair_sched_class;
   break;
@@ -3534,12 +3543,12 @@ recheck:
   policy = oldpolicy = p->policy;
  else if (policy != SCHED_FIFO && policy != SCHED_RR &&
    policy != SCHED_NORMAL && policy != SCHED_BATCH &&
-   policy != SCHED_IDLEPRIO)
+   policy != SCHED_ISO && policy != SCHED_IDLEPRIO)
   return -EINVAL;
  /*
   * Valid priorities for SCHED_FIFO and SCHED_RR are
   * 1..MAX_USER_RT_PRIO-1, valid priority for SCHED_NORMAL,
-  * SCHED_BATCH and SCHED_IDLEPRIO is 0.
+  * SCHED_BATCH, SCHED_ISO and SCHED_IDLEPRIO is 0.
   */
  if (param->sched_priority < 0 ||
      (p->mm && param->sched_priority > MAX_USER_RT_PRIO-1) ||
@@ -3570,6 +3579,12 @@ recheck:
        param->sched_priority > rlim_rtprio)
```

```
     return -EPERM;
   }
+  /*
+   * Like positive nice levels, dont allow tasks to
+   * move out of SCHED_IDLEPRIO either:
+   */
+  if (p->policy == SCHED_IDLEPRIO && policy != SCHED_IDLEPRIO)
+   return -EPERM;

   /* can't change other user's priorities */
   if ((current->euid != p->euid) &&
@@ -3597,7 +3612,7 @@ recheck:
   spin_unlock_irqrestore(&p->pi_lock, flags);
   goto recheck;
  }
- on_rq = p->on_rq;
+ on_rq = p->se.on_rq;
  if (on_rq)
   deactivate_task(rq, p, 0);
  oldprio = p->prio;
@@ -4093,6 +4108,7 @@ asmlinkage long sys_sched_get_priority_m
   break;
  case SCHED_NORMAL:
  case SCHED_BATCH:
+ case SCHED_ISO:
  case SCHED_IDLEPRIO:
   ret = 0;
   break;
@@ -4118,6 +4134,7 @@ asmlinkage long sys_sched_get_priority_m
   break;
  case SCHED_NORMAL:
  case SCHED_BATCH:
+ case SCHED_ISO:
  case SCHED_IDLEPRIO:
   ret = 0;
  }
@@ -4249,7 +4266,7 @@ void __cpuinit init_idle(struct task_str
  unsigned long flags;

  __sched_fork(idle);
- idle->exec_start = sched_clock();
+ idle->se.exec_start = sched_clock();

  idle->prio = idle->normal_prio = MAX_PRIO;
  idle->cpus_allowed = cpumask_of_cpu(cpu);
@@ -4352,7 +4369,7 @@ EXPORT_SYMBOL_GPL(set_cpus_allowed);
 static int __migrate_task(struct task_struct *p, int src_cpu, int dest_cpu)
 {
```

```
  struct rq *rq_dest, *rq_src;
- int ret = 0;
+ int ret = 0, on_rq;

  if (unlikely(cpu_is_offline(dest_cpu)))
   return ret;
@@ -4368,9 +4385,11 @@ static int __migrate_task(struct task_st
  if (!cpu_isset(dest_cpu, p->cpus_allowed))
   goto out;

- set_task_cpu(p, dest_cpu);
- if (p->on_rq) {
+ on_rq = p->se.on_rq;
+ if (on_rq)
   deactivate_task(rq_src, p, 0);
+ set_task_cpu(p, dest_cpu);
+ if (on_rq) {
   activate_task(rq_dest, p, 0);
   check_preempt_curr(rq_dest, p);
  }
@@ -5752,11 +5771,11 @@ void __init sched_init(void)
   spin_lock_init(&rq->lock);
   lockdep_set_class(&rq->lock, &rq->rq_lock_key);
   rq->nr_running = 0;
-  rq->tasks_timeline = RB_ROOT;
-  rq->clock = rq->fair_clock = 1;
+  rq->lrq.tasks_timeline = RB_ROOT;
+  rq->clock = rq->lrq.fair_clock = 1;

   for (j = 0; j < CPU_LOAD_IDX_MAX; j++)
-   rq->cpu_load[j] = 0;
+   rq->lrq.cpu_load[j] = 0;
 #ifdef CONFIG_SMP
   rq->sd = NULL;
   rq->active_balance = 0;
@@ -5836,15 +5855,15 @@ void normalize_rt_tasks(void)

  read_lock_irq(&tasklist_lock);
  do_each_thread(g, p) {
-  p->fair_key = 0;
-  p->wait_runtime = 0;
-  p->wait_start_fair = 0;
-  p->wait_start = 0;
-  p->exec_start = 0;
-  p->sleep_start = 0;
-  p->sleep_start_fair = 0;
-  p->block_start = 0;
-  task_rq(p)->fair_clock = 0;
```

```
+  p->se.fair_key = 0;
+  p->se.wait_runtime = 0;
+  p->se.wait_start_fair = 0;
+  p->se.wait_start = 0;
+  p->se.exec_start = 0;
+  p->se.sleep_start = 0;
+  p->se.sleep_start_fair = 0;
+  p->se.block_start = 0;
+  task_rq(p)->lrq.fair_clock = 0;
   task_rq(p)->clock = 0;

   if (!rt_task(p)) {
@@ -5867,7 +5886,7 @@ void normalize_rt_tasks(void)
    goto out_unlock;
 #endif

-  on_rq = p->on_rq;
+  on_rq = p->se.on_rq;
   if (on_rq)
    deactivate_task(task_rq(p), p, 0);
   __setscheduler(rq, p, SCHED_NORMAL, 0);
Index: linux/kernel/sched_debug.c
===================================================================
--- linux.orig/kernel/sched_debug.c
+++ linux/kernel/sched_debug.c
@@ -40,19 +40,19 @@ print_task(struct seq_file *m, struct rq
  SEQ_printf(m, "%15s %5d %15Ld %13Ld %13Ld %9Ld %5d "
        "%15Ld %15Ld %15Ld %15Ld %15Ld\n",
   p->comm, p->pid,
-  (long long)p->fair_key,
-  (long long)(p->fair_key - rq->fair_clock),
-  (long long)p->wait_runtime,
+  (long long)p->se.fair_key,
+  (long long)(p->se.fair_key - rq->lrq.fair_clock),
+  (long long)p->se.wait_runtime,
   (long long)p->nr_switches,
   p->prio,
-  (long long)p->sum_exec_runtime,
-  (long long)p->sum_wait_runtime,
-  (long long)p->sum_sleep_runtime,
-  (long long)p->wait_runtime_overruns,
-  (long long)p->wait_runtime_underruns);
+  (long long)p->se.sum_exec_runtime,
+  (long long)p->se.sum_wait_runtime,
+  (long long)p->se.sum_sleep_runtime,
+  (long long)p->se.wait_runtime_overruns,
+  (long long)p->se.wait_runtime_underruns);
 }
```

```
-static void print_rq(struct seq_file *m, struct rq *rq, u64 now)
+static void print_rq(struct seq_file *m, struct rq *rq, int rq_cpu, u64 now)
 {
  struct task_struct *g, *p;

@@ -70,7 +70,7 @@ static void print_rq(struct seq_file *m,
  read_lock_irq(&tasklist_lock);

  do_each_thread(g, p) {
-  if (!p->on_rq)
+  if (!p->se.on_rq || task_cpu(p) != rq_cpu)
    continue;

   print_task(m, rq, p, now);
@@ -87,10 +87,10 @@ static void print_rq_runtime_sum(struct
  unsigned long flags;

  spin_lock_irqsave(&rq->lock, flags);
- curr = first_fair(rq);
+ curr = first_fair(&rq->lrq);
  while (curr) {
-  p = rb_entry(curr, struct task_struct, run_node);
-  wait_runtime_rq_sum += p->wait_runtime;
+  p = rb_entry(curr, struct task_struct, se.run_node);
+  wait_runtime_rq_sum += p->se.wait_runtime;

   curr = rb_next(curr);
  }
@@ -109,9 +109,9 @@ static void print_cpu(struct seq_file *m
  SEQ_printf(m, "  .%-22s: %Ld\n", #x, (long long)(rq->x))

  P(nr_running);
- P(raw_weighted_load);
+ P(lrq.raw_weighted_load);
  P(nr_switches);
- P(nr_load_updates);
+ P(lrq.nr_load_updates);
  P(nr_uninterruptible);
  SEQ_printf(m, "  .%-22s: %lu\n", "jiffies", jiffies);
  P(next_balance);
@@ -122,22 +122,22 @@ static void print_cpu(struct seq_file *m
  P(clock_overflows);
  P(clock_unstable_events);
  P(clock_max_delta);
- P(fair_clock);
- P(delta_fair_clock);
- P(exec_clock);
```

```
- P(delta_exec_clock);
- P(wait_runtime);
- P(wait_runtime_overruns);
- P(wait_runtime_underruns);
- P(cpu_load[0]);
- P(cpu_load[1]);
- P(cpu_load[2]);
- P(cpu_load[3]);
- P(cpu_load[4]);
+ P(lrq.fair_clock);
+ P(lrq.delta_fair_clock);
+ P(lrq.exec_clock);
+ P(lrq.delta_exec_clock);
+ P(lrq.wait_runtime);
+ P(lrq.wait_runtime_overruns);
+ P(lrq.wait_runtime_underruns);
+ P(lrq.cpu_load[0]);
+ P(lrq.cpu_load[1]);
+ P(lrq.cpu_load[2]);
+ P(lrq.cpu_load[3]);
+ P(lrq.cpu_load[4]);
 #undef P
  print_rq_runtime_sum(m, rq);

- print_rq(m, rq, now);
+ print_rq(m, rq, cpu, now);
 }

 static int sched_debug_show(struct seq_file *m, void *v)
@@ -205,21 +205,21 @@ void proc_sched_show_task(struct task_st
 #define P(F) \
  SEQ_printf(m, "%-25s:%20Ld\n", #F, (long long)p->F)

- P(wait_start);
- P(wait_start_fair);
- P(exec_start);
- P(sleep_start);
- P(sleep_start_fair);
- P(block_start);
- P(sleep_max);
- P(block_max);
- P(exec_max);
- P(wait_max);
- P(wait_runtime);
- P(wait_runtime_overruns);
- P(wait_runtime_underruns);
- P(sum_exec_runtime);
- P(load_weight);
```

```
+ P(se.wait_start);
+ P(se.wait_start_fair);
+ P(se.exec_start);
+ P(se.sleep_start);
+ P(se.sleep_start_fair);
+ P(se.block_start);
+ P(se.sleep_max);
+ P(se.block_max);
+ P(se.exec_max);
+ P(se.wait_max);
+ P(se.wait_runtime);
+ P(se.wait_runtime_overruns);
+ P(se.wait_runtime_underruns);
+ P(se.sum_exec_runtime);
+ P(se.load_weight);
  P(policy);
  P(prio);
 #undef P
@@ -235,7 +235,7 @@ void proc_sched_show_task(struct task_st

 void proc_sched_set_task(struct task_struct *p)
 {
- p->sleep_max = p->block_max = p->exec_max = p->wait_max = 0;
- p->wait_runtime_overruns = p->wait_runtime_underruns = 0;
- p->sum_exec_runtime = 0;
+ p->se.sleep_max = p->se.block_max = p->se.exec_max = p->se.wait_max = 0;
+ p->se.wait_runtime_overruns = p->se.wait_runtime_underruns = 0;
+ p->se.sum_exec_runtime = 0;
 }
Index: linux/kernel/sched_fair.c
===================================================================
--- linux.orig/kernel/sched_fair.c
+++ linux/kernel/sched_fair.c
@@ -38,22 +38,57 @@ unsigned int sysctl_sched_batch_wakeup_g
  */
 unsigned int sysctl_sched_runtime_limit __read_mostly;

-unsigned int sysctl_sched_features __read_mostly = 1 | 2 | 4 | 8 | 0 | 0;
+unsigned int sysctl_sched_features __read_mostly = 0 | 2 | 4 | 8 | 0 | 0;

 extern struct sched_class fair_sched_class;

+/**************************************************************************/
+/*           BEGIN : CFS operations on generic schedulable entities       */
+/**************************************************************************/
+
+static inline struct rq *lrq_rq(struct lrq *lrq)
+{
```

```
+ return container_of(lrq, struct rq, lrq);
+}
+
+static inline struct sched_entity *lrq_curr(struct lrq *lrq)
+{
+ struct rq *rq = lrq_rq(lrq);
+ struct sched_entity *se = NULL;
+
+ if (rq->curr->sched_class == &fair_sched_class)
+  se = &rq->curr->se;
+
+ return se;
+}
+
+static long lrq_nr_running(struct lrq *lrq)
+{
+ struct rq *rq = lrq_rq(lrq);
+
+ return rq->nr_running;
+}
+
+#define entity_is_task(se) 1
+
+static inline struct task_struct *entity_to_task(struct sched_entity *se)
+{
+ return container_of(se, struct task_struct, se);
+}
+
+
 /**************************************************************/
 /* Scheduling class tree data structure manipulation methods:
  */

 /*
- * Enqueue a task into the rb-tree:
+ * Enqueue a entity into the rb-tree:
  */
-static inline void __enqueue_task_fair(struct rq *rq, struct task_struct *p)
+static inline void __enqueue_entity(struct lrq *lrq, struct sched_entity *p)
 {
- struct rb_node **link = &rq->tasks_timeline.rb_node;
+ struct rb_node **link = &lrq->tasks_timeline.rb_node;
  struct rb_node *parent = NULL;
- struct task_struct *entry;
+ struct sched_entity *entry;
  s64 key = p->fair_key;
  int leftmost = 1;
```

```
@@ -62,7 +97,7 @@ static inline void __enqueue_task_fair(s
   */
  while (*link) {
   parent = *link;
-  entry = rb_entry(parent, struct task_struct, run_node);
+  entry = rb_entry(parent, struct sched_entity, run_node);
   /*
    * We dont care about collisions. Nodes with
    * the same key stay together.
@@ -80,31 +115,31 @@ static inline void __enqueue_task_fair(s
   * used):
   */
  if (leftmost)
-  rq->rb_leftmost = &p->run_node;
+  lrq->rb_leftmost = &p->run_node;

  rb_link_node(&p->run_node, parent, link);
- rb_insert_color(&p->run_node, &rq->tasks_timeline);
+ rb_insert_color(&p->run_node, &lrq->tasks_timeline);
 }

-static inline void __dequeue_task_fair(struct rq *rq, struct task_struct *p)
+static inline void __dequeue_entity(struct lrq *lrq, struct sched_entity *p)
 {
- if (rq->rb_leftmost == &p->run_node)
-  rq->rb_leftmost = NULL;
- rb_erase(&p->run_node, &rq->tasks_timeline);
+ if (lrq->rb_leftmost == &p->run_node)
+  lrq->rb_leftmost = NULL;
+ rb_erase(&p->run_node, &lrq->tasks_timeline);
 }

-static inline struct rb_node * first_fair(struct rq *rq)
+static inline struct rb_node * first_fair(struct lrq *lrq)
 {
- if (rq->rb_leftmost)
-  return rq->rb_leftmost;
+ if (lrq->rb_leftmost)
+  return lrq->rb_leftmost;
  /* Cache the value returned by rb_first() */
- rq->rb_leftmost = rb_first(&rq->tasks_timeline);
- return rq->rb_leftmost;
+ lrq->rb_leftmost = rb_first(&lrq->tasks_timeline);
+ return lrq->rb_leftmost;
 }

-static struct task_struct * __pick_next_task_fair(struct rq *rq)
+static struct sched_entity * __pick_next_entity(struct lrq *lrq)
```

```
 {
- return rb_entry(first_fair(rq), struct task_struct, run_node);
+ return rb_entry(first_fair(lrq), struct sched_entity, run_node);
 }

 /**************************************************************/
@@ -115,8 +150,8 @@ static struct task_struct * __pick_next_
  * We rescale the rescheduling granularity of tasks according to their
  * nice level, but only linearly, not exponentially:
  */
-static u64
-niced_granularity(struct task_struct *curr, unsigned long granularity)
+static s64
+niced_granularity(struct sched_entity *curr, unsigned long granularity)
 {
  /*
   * Negative nice levels get the same granularity as nice-0:
@@ -130,7 +165,7 @@ niced_granularity(struct task_struct *cu
  return curr->load_weight * (s64)(granularity / NICE_0_LOAD);
 }

-static void limit_wait_runtime(struct rq *rq, struct task_struct *p)
+static void limit_wait_runtime(struct lrq *lrq, struct sched_entity *p)
 {
  s64 limit = sysctl_sched_runtime_limit;

@@ -141,27 +176,28 @@ static void limit_wait_runtime(struct rq
  if (p->wait_runtime > limit) {
   p->wait_runtime = limit;
   p->wait_runtime_overruns++;
-  rq->wait_runtime_overruns++;
+  lrq->wait_runtime_overruns++;
  }
  if (p->wait_runtime < -limit) {
   p->wait_runtime = -limit;
   p->wait_runtime_underruns++;
-  rq->wait_runtime_underruns++;
+  lrq->wait_runtime_underruns++;
  }
 }

-static void __add_wait_runtime(struct rq *rq, struct task_struct *p, s64 delta)
+static void
+__add_wait_runtime(struct lrq *lrq, struct sched_entity *p, s64 delta)
 {
  p->wait_runtime += delta;
  p->sum_wait_runtime += delta;
- limit_wait_runtime(rq, p);
```

```
+ limit_wait_runtime(lrq, p);
 }

-static void add_wait_runtime(struct rq *rq, struct task_struct *p, s64 delta)
+static void add_wait_runtime(struct lrq *lrq, struct sched_entity *p, s64 delta)
 {
- rq->wait_runtime -= p->wait_runtime;
- __add_wait_runtime(rq, p, delta);
- rq->wait_runtime += p->wait_runtime;
+ lrq->wait_runtime -= p->wait_runtime;
+ __add_wait_runtime(lrq, p, delta);
+ lrq->wait_runtime += p->wait_runtime;
 }

 static s64 div64_s(s64 divident, unsigned long divisor)
@@ -183,13 +219,15 @@ static s64 div64_s(s64 divident, unsigne
  * Update the current task's runtime statistics. Skip current tasks that
  * are not in our scheduling class.
  */
-static inline void update_curr(struct rq *rq, u64 now)
+static inline void update_curr(struct lrq *lrq, u64 now)
 {
- unsigned long load = rq->raw_weighted_load;
+ unsigned long load = lrq->raw_weighted_load;
  u64 delta_exec, delta_fair, delta_mine;
- struct task_struct *curr = rq->curr;
+ struct sched_entity *curr = lrq_curr(lrq);
+ struct rq *rq = lrq_rq(lrq);
+ struct task_struct *curtask = rq->curr;

- if (curr->sched_class != &fair_sched_class || curr == rq->idle || !load)
+ if (!curr || curtask == rq->idle || !load)
   return;
  /*
   * Get the amount of time the current task was running
@@ -203,29 +241,29 @@ static inline void update_curr(struct rq

  curr->sum_exec_runtime += delta_exec;
  curr->exec_start = now;
- rq->exec_clock += delta_exec;
+ lrq->exec_clock += delta_exec;

  delta_fair = delta_exec * NICE_0_LOAD;
  delta_fair += load >> 1; /* rounding */
  do_div(delta_fair, load);

  /* Load-balancing accounting. */
- rq->delta_fair_clock += delta_fair;
```

```
- rq->delta_exec_clock += delta_exec;
+ lrq->delta_fair_clock += delta_fair;
+ lrq->delta_exec_clock += delta_exec;

  /*
   * Task already marked for preemption, do not burden
   * it with the cost of not having left the CPU yet:
   */
  if (unlikely(sysctl_sched_features & 1))
-  if (unlikely(test_tsk_thread_flag(curr, TIF_NEED_RESCHED)))
+  if (unlikely(test_tsk_thread_flag(curtask, TIF_NEED_RESCHED)))
    return;

  delta_mine = delta_exec * curr->load_weight;
  delta_mine += load >> 1; /* rounding */
  do_div(delta_mine, load);

- rq->fair_clock += delta_fair;
+ lrq->fair_clock += delta_fair;
  /*
   * We executed delta_exec amount of time on the CPU,
   * but we were only entitled to delta_mine amount of
@@ -233,13 +271,13 @@ static inline void update_curr(struct rq
   * the two values are equal)
   * [Note: delta_mine - delta_exec is negative]:
   */
- add_wait_runtime(rq, curr, delta_mine - delta_exec);
+ add_wait_runtime(lrq, curr, delta_mine - delta_exec);
 }

 static inline void
-update_stats_wait_start(struct rq *rq, struct task_struct *p, u64 now)
+update_stats_wait_start(struct lrq *lrq, struct sched_entity *p, u64 now)
 {
- p->wait_start_fair = rq->fair_clock;
+ p->wait_start_fair = lrq->fair_clock;
  p->wait_start = now;
 }

@@ -247,7 +285,7 @@ update_stats_wait_start(struct rq *rq, s
  * Task is being enqueued - update stats:
  */
 static inline void
-update_stats_enqueue(struct rq *rq, struct task_struct *p, u64 now)
+update_stats_enqueue(struct lrq *lrq, struct sched_entity *p, u64 now)
 {
  s64 key;
```

```
@@ -255,12 +293,12 @@ update_stats_enqueue(struct rq *rq, stru
   * Are we enqueueing a waiting task? (for current tasks
   * a dequeue/enqueue event is a NOP)
   */
- if (p != rq->curr)
-  update_stats_wait_start(rq, p, now);
+ if (p != lrq_curr(lrq))
+  update_stats_wait_start(lrq, p, now);
  /*
   * Update the key:
   */
- key = rq->fair_clock;
+ key = lrq->fair_clock;

  /*
   * Optimize the common nice 0 case:
@@ -269,9 +307,11 @@ update_stats_enqueue(struct rq *rq, stru
   key -= p->wait_runtime;
  else {
   if (p->wait_runtime < 0)
-   key -= div64_s(p->wait_runtime * NICE_0_LOAD, p->load_weight);
+   key -= div64_s(p->wait_runtime * NICE_0_LOAD,
+         p->load_weight);
   else
-   key -= div64_s(p->wait_runtime * p->load_weight, NICE_0_LOAD);
+   key -= div64_s(p->wait_runtime * p->load_weight,
+          NICE_0_LOAD);
  }

  p->fair_key = key;
@@ -281,7 +321,7 @@ update_stats_enqueue(struct rq *rq, stru
  * Note: must be called with a freshly updated rq->fair_clock.
  */
 static inline void
-update_stats_wait_end(struct rq *rq, struct task_struct *p, u64 now)
+update_stats_wait_end(struct lrq *lrq, struct sched_entity *p, u64 now)
 {
  s64 delta_fair, delta_wait;

@@ -290,12 +330,12 @@ update_stats_wait_end(struct rq *rq, str
   p->wait_max = delta_wait;

  if (p->wait_start_fair) {
-  delta_fair = rq->fair_clock - p->wait_start_fair;
+  delta_fair = lrq->fair_clock - p->wait_start_fair;

   if (unlikely(p->load_weight != NICE_0_LOAD))
    delta_fair = div64_s(delta_fair * p->load_weight,
```

```
        NICE_0_LOAD);
-  add_wait_runtime(rq, p, delta_fair);
+  add_wait_runtime(lrq, p, delta_fair);
  }

  p->wait_start_fair = 0;
@@ -303,22 +343,22 @@ update_stats_wait_end(struct rq *rq, str
 }

 static inline void
-update_stats_dequeue(struct rq *rq, struct task_struct *p, u64 now)
+update_stats_dequeue(struct lrq *lrq, struct sched_entity *p, u64 now)
 {
- update_curr(rq, now);
+ update_curr(lrq, now);
  /*
   * Mark the end of the wait period if dequeueing a
   * waiting task:
   */
- if (p != rq->curr)
-  update_stats_wait_end(rq, p, now);
+ if (p != lrq_curr(lrq))
+  update_stats_wait_end(lrq, p, now);
 }

 /*
  * We are picking a new current task - update its stats:
  */
 static inline void
-update_stats_curr_start(struct rq *rq, struct task_struct *p, u64 now)
+update_stats_curr_start(struct lrq *lrq, struct sched_entity *p, u64 now)
 {
  /*
   * We are starting a new run period:
@@ -330,7 +370,7 @@ update_stats_curr_start(struct rq *rq, s
  * We are descheduling a task - update its stats:
  */
 static inline void
-update_stats_curr_end(struct rq *rq, struct task_struct *p, u64 now)
+update_stats_curr_end(struct lrq *lrq, struct sched_entity *p, u64 now)
 {
  p->exec_start = 0;
 }
@@ -345,50 +385,53 @@ update_stats_curr_end(struct rq *rq, str
  * manner we move the fair clock back by a proportional
  * amount of the new wait_runtime this task adds to the pool.
  */
-static void distribute_fair_add(struct rq *rq, s64 delta)
```

```
+static void distribute_fair_add(struct lrq *lrq, s64 delta)
 {
- struct task_struct *curr = rq->curr;
+ struct sched_entity *curr = lrq_curr(lrq);
  s64 delta_fair = 0;

  if (!(sysctl_sched_features & 2))
   return;

- if (rq->nr_running) {
-  delta_fair = div64_s(delta, rq->nr_running);
+ if (lrq_nr_running(lrq)) {
+  delta_fair = div64_s(delta, lrq_nr_running(lrq));
   /*
    * The currently running task's next wait_runtime value does
    * not depend on the fair_clock, so fix it up explicitly:
    */
-   if (curr->sched_class == &fair_sched_class)
-   add_wait_runtime(rq, curr, -delta_fair);
+   if (curr)
+   add_wait_runtime(lrq, curr, -delta_fair);
  }
- rq->fair_clock -= delta_fair;
+ lrq->fair_clock -= delta_fair;
 }

 /**************************************************************/
 /* Scheduling class queueing methods:
  */

-static void enqueue_sleeper(struct rq *rq, struct task_struct *p)
+static void enqueue_sleeper(struct lrq *lrq, struct sched_entity *p)
 {
- unsigned long load = rq->raw_weighted_load;
+ unsigned long load = lrq->raw_weighted_load;
  s64 delta_fair, prev_runtime;
+ struct task_struct *tsk = entity_to_task(p);

- if (p->policy == SCHED_BATCH || !(sysctl_sched_features & 4))
+ if ((entity_is_task(p) && tsk->policy == SCHED_BATCH) ||
+       !(sysctl_sched_features & 4))
   goto out;

- delta_fair = rq->fair_clock - p->sleep_start_fair;
+ delta_fair = lrq->fair_clock - p->sleep_start_fair;

  /*
   * Fix up delta_fair with the effect of us running
```

```
   * during the whole sleep period:
   */
  if (!(sysctl_sched_features & 32))
-  delta_fair = div64_s(delta_fair * load, load + p->load_weight);
+  delta_fair = div64_s(delta_fair * load,
+       load + p->load_weight);
  delta_fair = div64_s(delta_fair * p->load_weight, NICE_0_LOAD);

  prev_runtime = p->wait_runtime;
- __add_wait_runtime(rq, p, delta_fair);
+ __add_wait_runtime(lrq, p, delta_fair);
  delta_fair = p->wait_runtime - prev_runtime;

  /*
@@ -396,28 +439,23 @@ static void enqueue_sleeper(struct rq *r
   * amount of the new wait_runtime this task adds to
   * the 'pool':
   */
- distribute_fair_add(rq, delta_fair);
+ distribute_fair_add(lrq, delta_fair);

 out:
- rq->wait_runtime += p->wait_runtime;
+ lrq->wait_runtime += p->wait_runtime;

  p->sleep_start_fair = 0;
 }

-/*
- * The enqueue_task method is called before nr_running is
- * increased. Here we update the fair scheduling stats and
- * then put the task into the rbtree:
- */
 static void
-enqueue_task_fair(struct rq *rq, struct task_struct *p, int wakeup, u64 now)
+enqueue_entity(struct lrq *lrq, struct sched_entity *p, int wakeup, u64 now)
 {
  u64 delta = 0;

  /*
   * Update the fair clock.
   */
- update_curr(rq, now);
+ update_curr(lrq, now);

  if (wakeup) {
   if (p->sleep_start) {
@@ -443,10 +481,152 @@ enqueue_task_fair(struct rq *rq, struct
```

```
   p->sum_sleep_runtime += delta;

   if (p->sleep_start_fair)
-   enqueue_sleeper(rq, p);
+    enqueue_sleeper(lrq, p);
+ }
+ update_stats_enqueue(lrq, p, now);
+ __enqueue_entity(lrq, p);
+}
+
+static void
+dequeue_entity(struct lrq *lrq, struct sched_entity *p, int sleep, u64 now)
+{
+ update_stats_dequeue(lrq, p, now);
+ if (sleep) {
+  if (entity_is_task(p)) {
+   struct task_struct *tsk = entity_to_task(p);
+
+   if (tsk->state & TASK_INTERRUPTIBLE)
+    p->sleep_start = now;
+   if (tsk->state & TASK_UNINTERRUPTIBLE)
+    p->block_start = now;
+  }
+  p->sleep_start_fair = lrq->fair_clock;
+  lrq->wait_runtime -= p->wait_runtime;
+ }
+ __dequeue_entity(lrq, p);
+}
+
+/*
+ * Preempt the current task with a newly woken task if needed:
+ */
+static inline void
+__check_preempt_curr_fair(struct lrq *lrq, struct sched_entity *p,
+     struct sched_entity *curr, unsigned long granularity)
+{
+ s64 __delta = curr->fair_key - p->fair_key;
+
+ /*
+  * Take scheduling granularity into account - do not
+  * preempt the current task unless the best task has
+  * a larger than sched_granularity fairness advantage:
+  */
+ if (__delta > niced_granularity(curr, granularity))
+  resched_task(lrq_rq(lrq)->curr);
+}
+
+static struct sched_entity * pick_next_entity(struct lrq *lrq, u64 now)
```

```
+{
+ struct sched_entity *p = __pick_next_entity(lrq);
+
+ /*
+  * 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. (note, here we rely on pick_next_task() having
+  * done a put_prev_task_fair() shortly before this, which
+  * updated rq->fair_clock - used by update_stats_wait_end())
+  */
+ update_stats_wait_end(lrq, p, now);
+ update_stats_curr_start(lrq, p, now);
+
+ return p;
+}
+
+static void put_prev_entity(struct lrq *lrq, struct sched_entity *prev, u64 now)
+{
+ /*
+  * If the task is still waiting for the CPU (it just got
+  * preempted), update its position within the tree and
+  * start the wait period:
+  */
+ if ((sysctl_sched_features & 16) && entity_is_task(prev))  {
+  struct task_struct *prevtask = entity_to_task(prev);
+
+  if (prev->on_rq &&
+   test_tsk_thread_flag(prevtask, TIF_NEED_RESCHED)) {
+
+   dequeue_entity(lrq, prev, 0, now);
+   prev->on_rq = 0;
+   enqueue_entity(lrq, prev, 0, now);
+   prev->on_rq = 1;
+  } else
+   update_curr(lrq, now);
+ } else {
+  update_curr(lrq, now);
+ }
+
+ update_stats_curr_end(lrq, prev, now);
+
+ if (prev->on_rq)
+  update_stats_wait_start(lrq, prev, now);
+}
+
+static void entity_tick(struct lrq *lrq, struct sched_entity *curr)
+{
+ struct sched_entity *next;
```

```
+ struct rq *rq = lrq_rq(lrq);
+ u64 now = __rq_clock(rq);
+
+ /*
+  * Dequeue and enqueue the task to update its
+  * position within the tree:
+  */
+ dequeue_entity(lrq, curr, 0, now);
+ curr->on_rq = 0;
+ enqueue_entity(lrq, curr, 0, now);
+ curr->on_rq = 1;
+
+ /*
+  * Reschedule if another task tops the current one.
+  */
+ next = __pick_next_entity(lrq);
+ if (next == curr)
+  return;
+
+ if (entity_is_task(curr)) {
+  struct task_struct *curtask = entity_to_task(curr),
+       *nexttask = entity_to_task(next);
+
+  if ((curtask == rq->idle) || (rt_prio(nexttask->prio) &&
+     (nexttask->prio < curtask->prio))) {
+   resched_task(curtask);
+   return;
+  }
  }
- update_stats_enqueue(rq, p, now);
- __enqueue_task_fair(rq, p);
+ __check_preempt_curr_fair(lrq, next, curr, sysctl_sched_granularity);
+}
+
+
+/**************************************************************************/
+/*                    BEGIN : CFS operations on tasks                     */
+/**************************************************************************/
+
+static inline struct lrq *task_lrq(struct task_struct *p)
+{
+ return &task_rq(p)->lrq;
+}
+
+/*
+ * The enqueue_task method is called before nr_running is
+ * increased. Here we update the fair scheduling stats and
+ * then put the task into the rbtree:
```

```
+ */
+static void
+enqueue_task_fair(struct rq *rq, struct task_struct *p, int wakeup, u64 now)
+{
+ struct lrq *lrq = task_lrq(p);
+ struct sched_entity *se = &p->se;
+
+ enqueue_entity(lrq, se, wakeup, now);
 }

 /*
@@ -457,16 +637,10 @@ enqueue_task_fair(struct rq *rq, struct
 static void
 dequeue_task_fair(struct rq *rq, struct task_struct *p, int sleep, u64 now)
 {
- update_stats_dequeue(rq, p, now);
- if (sleep) {
-  if (p->state & TASK_INTERRUPTIBLE)
-   p->sleep_start = now;
-  if (p->state & TASK_UNINTERRUPTIBLE)
-   p->block_start = now;
-  p->sleep_start_fair = rq->fair_clock;
-  rq->wait_runtime -= p->wait_runtime;
- }
- __dequeue_task_fair(rq, p);
+ struct lrq *lrq = task_lrq(p);
+ struct sched_entity *se = &p->se;
+
+ dequeue_entity(lrq, se, sleep, now);
 }

 /*
@@ -479,16 +653,18 @@ yield_task_fair(struct rq *rq, struct ta
 {
  struct task_struct *p_next;
  u64 now;
+ struct lrq *lrq = task_lrq(p);
+ struct sched_entity *se = &p->se;

  now = __rq_clock(rq);
  /*
   * Dequeue and enqueue the task to update its
   * position within the tree:
   */
- dequeue_task_fair(rq, p, 0, now);
- p->on_rq = 0;
- enqueue_task_fair(rq, p, 0, now);
- p->on_rq = 1;
```

```
+ dequeue_entity(lrq, se, 0, now);
+ se->on_rq = 0;
+ enqueue_entity(lrq, se, 0, now);
+ se->on_rq = 1;

  /*
   * yield-to support: if we are on the same runqueue then
@@ -496,39 +672,23 @@ yield_task_fair(struct rq *rq, struct ta
   */
  if (p_to && rq == task_rq(p_to) &&
    p_to->sched_class == &fair_sched_class
-   && p->wait_runtime > 0) {
+   && p->se.wait_runtime > 0) {

-  s64 delta = p->wait_runtime >> 1;
+  s64 delta = p->se.wait_runtime >> 1;

-  __add_wait_runtime(rq, p_to, delta);
-  __add_wait_runtime(rq, p, -delta);
+  __add_wait_runtime(lrq, &p_to->se, delta);
+  __add_wait_runtime(lrq, &p->se, -delta);
  }

  /*
   * Reschedule if another task tops the current one.
   */
- p_next = __pick_next_task_fair(rq);
+ se = __pick_next_entity(lrq);
+ p_next = entity_to_task(se);
  if (p_next != p)
   resched_task(p);
 }

-/*
- * Preempt the current task with a newly woken task if needed:
- */
-static inline void
-__check_preempt_curr_fair(struct rq *rq, struct task_struct *p,
-     struct task_struct *curr, unsigned long granularity)
-{
- s64 __delta = curr->fair_key - p->fair_key;
-
- /*
-  * Take scheduling granularity into account - do not
-  * preempt the current task unless the best task has
-  * a larger than sched_granularity fairness advantage:
-  */
- if (__delta > niced_granularity(curr, granularity))
```

```
-  resched_task(curr);
-}

 /*
  * Preempt the current task with a newly woken task if needed:
@@ -536,12 +696,13 @@ __check_preempt_curr_fair(struct rq *rq,
 static void check_preempt_curr_fair(struct rq *rq, struct task_struct *p)
 {
  struct task_struct *curr = rq->curr;
+ struct lrq *lrq = task_lrq(curr);
  unsigned long granularity;

  if ((curr == rq->idle) || rt_prio(p->prio)) {
   if (sysctl_sched_features & 8) {
    if (rt_prio(p->prio))
-    update_curr(rq, rq_clock(rq));
+    update_curr(lrq, rq_clock(rq));
   }
   resched_task(curr);
  } else {
@@ -552,25 +713,18 @@ static void check_preempt_curr_fair(stru
   if (unlikely(p->policy == SCHED_BATCH))
    granularity = sysctl_sched_batch_wakeup_granularity;

-  __check_preempt_curr_fair(rq, p, curr, granularity);
+  __check_preempt_curr_fair(lrq, &p->se, &curr->se, granularity);
  }
 }

 static struct task_struct * pick_next_task_fair(struct rq *rq, u64 now)
 {
- struct task_struct *p = __pick_next_task_fair(rq);
+ struct lrq *lrq = &rq->lrq;
+ 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. (note, here we rely on pick_next_task() having
-  * done a put_prev_task_fair() shortly before this, which
-  * updated rq->fair_clock - used by update_stats_wait_end())
-  */
- update_stats_wait_end(rq, p, now);
- update_stats_curr_start(rq, p, now);
+ se = pick_next_entity(lrq, now);

- return p;
+ return entity_to_task(se);
```

```
 }

 /*
@@ -578,32 +732,13 @@ static struct task_struct * pick_next_ta
  */
 static void put_prev_task_fair(struct rq *rq, struct task_struct *prev, u64 now)
 {
+ struct lrq *lrq = task_lrq(prev);
+ struct sched_entity *se = &prev->se;
+
  if (prev == rq->idle)
   return;

- /*
-  * If the task is still waiting for the CPU (it just got
-  * preempted), update its position within the tree and
-  * start the wait period:
-  */
- if (sysctl_sched_features & 16)  {
-  if (prev->on_rq &&
-   test_tsk_thread_flag(prev, TIF_NEED_RESCHED)) {
-
-   dequeue_task_fair(rq, prev, 0, now);
-   prev->on_rq = 0;
-   enqueue_task_fair(rq, prev, 0, now);
-   prev->on_rq = 1;
-  } else
-   update_curr(rq, now);
- } else {
-  update_curr(rq, now);
- }
-
- update_stats_curr_end(rq, prev, now);
-
- if (prev->on_rq)
-  update_stats_wait_start(rq, prev, now);
+ put_prev_entity(lrq, se, now);
 }

 /*************************************************************/
@@ -625,20 +760,20 @@ __load_balance_iterator(struct rq *rq, s
  if (!curr)
   return NULL;

- p = rb_entry(curr, struct task_struct, run_node);
- rq->rb_load_balance_curr = rb_next(curr);
+ p = rb_entry(curr, struct task_struct, se.run_node);
+ rq->lrq.rb_load_balance_curr = rb_next(curr);
```

```
  return p;
 }

 static struct task_struct * load_balance_start_fair(struct rq *rq)
 {
- return __load_balance_iterator(rq, first_fair(rq));
+ return __load_balance_iterator(rq, first_fair(&rq->lrq));
 }

 static struct task_struct * load_balance_next_fair(struct rq *rq)
 {
- return __load_balance_iterator(rq, rq->rb_load_balance_curr);
+ return __load_balance_iterator(rq, rq->lrq.rb_load_balance_curr);
 }

 /*
@@ -646,31 +781,10 @@ static struct task_struct * load_balance
  */
 static void task_tick_fair(struct rq *rq, struct task_struct *curr)
 {
- struct task_struct *next;
- u64 now = __rq_clock(rq);
-
- /*
-  * Dequeue and enqueue the task to update its
-  * position within the tree:
-  */
- dequeue_task_fair(rq, curr, 0, now);
- curr->on_rq = 0;
- enqueue_task_fair(rq, curr, 0, now);
- curr->on_rq = 1;
+ struct lrq *lrq = task_lrq(curr);
+ struct sched_entity *se = &curr->se;

- /*
-  * Reschedule if another task tops the current one.
-  */
- next = __pick_next_task_fair(rq);
- if (next == curr)
-  return;
-
- if ((curr == rq->idle) || (rt_prio(next->prio) &&
-     (next->prio < curr->prio)))
-  resched_task(curr);
- else
-  __check_preempt_curr_fair(rq, next, curr,
-       sysctl_sched_granularity);
```

```
+ entity_tick(lrq, se);
 }

 /*
@@ -682,29 +796,32 @@ static void task_tick_fair(struct rq *rq
  */
 static void task_new_fair(struct rq *rq, struct task_struct *p)
 {
+ struct lrq *lrq = task_lrq(p);
+ struct sched_entity *se = &p->se;
+
  sched_info_queued(p);
- update_stats_enqueue(rq, p, rq_clock(rq));
+ update_stats_enqueue(lrq, se, rq_clock(rq));
  /*
   * Child runs first: we let it run before the parent
   * until it reschedules once. We set up the key so that
   * it will preempt the parent:
   */
- p->fair_key = current->fair_key - niced_granularity(rq->curr,
+ p->se.fair_key = current->se.fair_key - niced_granularity(&rq->curr->se,
       sysctl_sched_granularity) - 1;
  /*
   * The first wait is dominated by the child-runs-first logic,
   * so do not credit it with that waiting time yet:
   */
- p->wait_start_fair = 0;
+ p->se.wait_start_fair = 0;

  /*
   * The statistical average of wait_runtime is about
   * -granularity/2, so initialize the task with that:
   */
-// p->wait_runtime = -(s64)(sysctl_sched_granularity / 2);
+// p->se.wait_runtime = -(s64)(sysctl_sched_granularity / 2);

- __enqueue_task_fair(rq, p);
- p->on_rq = 1;
+ __enqueue_entity(lrq, se);
+ p->se.on_rq = 1;
  inc_nr_running(p, rq);
 }

Index: linux/kernel/sched_rt.c
===================================================================
--- linux.orig/kernel/sched_rt.c
+++ linux/kernel/sched_rt.c
@@ -15,14 +15,14 @@ static inline void update_curr_rt(struct
```

```
  if (!has_rt_policy(curr))
   return;

- delta_exec = now - curr->exec_start;
+ delta_exec = now - curr->se.exec_start;
  if (unlikely((s64)delta_exec < 0))
   delta_exec = 0;
- if (unlikely(delta_exec > curr->exec_max))
-  curr->exec_max = delta_exec;
+ if (unlikely(delta_exec > curr->se.exec_max))
+  curr->se.exec_max = delta_exec;

- curr->sum_exec_runtime += delta_exec;
- curr->exec_start = now;
+ curr->se.sum_exec_runtime += delta_exec;
+ curr->se.exec_start = now;
 }

 static void
@@ -89,7 +89,7 @@ static struct task_struct * pick_next_ta
  queue = array->queue + idx;
  next = list_entry(queue->next, struct task_struct, run_list);

- next->exec_start = now;
+ next->se.exec_start = now;

  return next;
 }
@@ -97,7 +97,7 @@ static struct task_struct * pick_next_ta
 static void put_prev_task_rt(struct rq *rq, struct task_struct *p, u64 now)
 {
  update_curr_rt(rq, now);
- p->exec_start = 0;
+ p->se.exec_start = 0;
 }

 /*
```



## File Attachments

1) sched-cfs-v17-rc4.patch, downloaded 349 times