Subject: [PATCH v3 4/6] add a new scheduler hook for context switch
Posted by Glauber Costa on Wed, 30 May 2012 09:48:35 GMT
View Forum Message <> Reply to Message

To be able to count the number of switches per-cgroup, and merging
Paul's hint that it would be better to do it in fair.c and rt.c,
I am introducing a new write-side walk through a new scheduler hook,
called at every context switch away from a task (prev).

Read-side is greatly simplified, and as I'll show, the performance impact
does not seem huge. First, aside from the function call, this walk is
O(depth), which is not likely to be huge (if it is, the performance
impact is indeed bad - but one can argue this is a good punishment)

Also, this walk is likely to be cache-hot, since it is at most the very
same loop done by put_prev_task, except it loops depth - 1 instead of depth
times. This is specially important not to hurt tasks in the root cgroup,
that will pay just a branch.

I am introducing a new hook, because the existing ones didn't seem
appropriate. The main possibilities would be put_prev_task and
pick_next_task.

With pick_next_task, there are two main problems:

1) first, the loop is only cache hot in pick_next_task if tasks actually
belong to the same class and group as prev. Depending on the workload,
this is possibly unlikely.

2) This is vulnerable to wrongdoings in accountings when exiting to idle.
Consider two groups A and B with a following pattern: A exits to idle,
but after that, B is scheduled. B, on the other hand, always get back
to itself when it yields to idle. This means that the to-idle transition
in A is never noted.

So because of that, I do believe that prev is the right point of that.

However, put_prev_task is called many times from multiple places,
and the logic to differentiate a context switch from another kind of
put would make a mess out of it.

On a 4-way x86_64, hackbench -pipe 1 process 4000 shows the following results:
- units are seconds to complete the whole benchmark
- percentual stdev for easier assesment

Task sitting in the root cgroup:
Without patchset: 4.857700 (0.69 %)
With  patchset: 4.863700 (0.63 %)

Difference : 0.12 %

Task sitting in a 3-level depth cgroup:
Without patchset: 5.120867 (1.60 %)
With patchset: 5.113800 (0.41 %)
Difference : 0.13 %

Task sitting in a 30-level depth cgroup (totally crazy scenario):
Without patchset: 8.829385 (2.63 %)
With patchset: 9.975467 (2.80 %)
Difference : 12 %

For any sane use case, the user is unlikely to be nesting for much more than
3-levels. For that, and for the important case for most people, the difference
is inside the standard deviation and can be said to be negligible.

Although the patch does add a penalty, it only does that for
deeply nested scenarios (but those are already paying a 100 % penalty
against no-nesting even without the patchset!)

I hope this approach is acceptable.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Peter Zijlstra <a.p.zijlstra@chello.nl>
CC: Paul Turner <pjt@google.com>
---
include/linux/sched.h | 1 +
kernel/sched/core.c | 9 +++++++++
kernel/sched/fair.c | 15 +++++++++++++++
kernel/sched/rt.c | 15 +++++++++++++++
kernel/sched/sched.h | 3 +++
5 files changed, 43 insertions(+)

```
diff --git a/include/linux/sched.h b/include/linux/sched.h
index f45c0b2..d28d6ec 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1084,6 +1084,7 @@ struct sched_class {

  struct task_struct * (*pick_next_task) (struct rq *rq);
  void (*put_prev_task) (struct rq *rq, struct task_struct *p);
+ void (*context_switch) (struct rq *rq, struct task_struct *p);

 #ifdef CONFIG_SMP
  int  (*select_task_rq)(struct task_struct *p, int sd_flag, int flags);
diff --git a/kernel/sched/core.c b/kernel/sched/core.c
index 4c1d7e9..db4f2c3 100644
--- a/kernel/sched/core.c
```

```
+++ b/kernel/sched/core.c
@@ -1894,6 +1894,14 @@ fire_sched_out_preempt_notifiers(struct task_struct *curr,

 #endif /* CONFIG_PREEMPT_NOTIFIERS */

+static void sched_class_context_switch(struct rq *rq, struct task_struct *prev)
+{
+#if defined(CONFIG_FAIR_GROUP_SCHED) || defined(CONFIG_RT_GROUP_SCHED)
+ if (prev->sched_class->context_switch)
+  prev->sched_class->context_switch(rq, prev);
+#endif
+}
+
 /**
  * prepare_task_switch - prepare to switch tasks
  * @rq: the runqueue preparing to switch
@@ -1911,6 +1919,7 @@ static inline void
 prepare_task_switch(struct rq *rq, struct task_struct *prev,
       struct task_struct *next)
 {
+ sched_class_context_switch(rq, prev);
  sched_info_switch(prev, next);
  perf_event_task_sched_out(prev, next);
  fire_sched_out_preempt_notifiers(prev, next);
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 940e6d1..c26fe38 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -2993,6 +2993,20 @@ static struct task_struct *pick_next_task_fair(struct rq *rq)
  return p;
 }

+static void context_switch_fair(struct rq *rq, struct task_struct *p)
+{
+#ifdef CONFIG_FAIR_GROUP_SCHED
+ struct cfs_rq *cfs_rq;
+ struct sched_entity *se = &p->se;
+
+ while (se->parent) {
+  se = se->parent;
+  cfs_rq = group_cfs_rq(se);
+  cfs_rq->nr_switches++;
+ }
+#endif
+}
+
 /*
  * Account for a descheduled task:
```

```
  */
@@ -5255,6 +5269,7 @@ const struct sched_class fair_sched_class = {
  .check_preempt_curr = check_preempt_wakeup,

  .pick_next_task  = pick_next_task_fair,
+ .context_switch  = context_switch_fair,
  .put_prev_task  = put_prev_task_fair,

 #ifdef CONFIG_SMP
diff --git a/kernel/sched/rt.c b/kernel/sched/rt.c
index 30ee4e2..6f416e4 100644
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -1392,6 +1392,20 @@ static struct task_struct *pick_next_task_rt(struct rq *rq)
  return p;
 }

+static void context_switch_rt(struct rq *rq, struct task_struct *p)
+{
+#ifdef CONFIG_RT_GROUP_SCHED
+ struct sched_rt_entity *rt_se = &p->rt;
+ struct rt_rq *rt_rq;
+
+ while (rt_se->parent) {
+  rt_se = rt_se->parent;
+  rt_rq = group_rt_rq(rt_se);
+  rt_rq->rt_nr_switches++;
+ }
+#endif
+}
+
 static void put_prev_task_rt(struct rq *rq, struct task_struct *p)
 {
  update_curr_rt(rq);
@@ -2040,6 +2054,7 @@ const struct sched_class rt_sched_class = {
  .check_preempt_curr = check_preempt_curr_rt,

  .pick_next_task  = pick_next_task_rt,
+ .context_switch  = context_switch_rt,
  .put_prev_task  = put_prev_task_rt,

 #ifdef CONFIG_SMP
diff --git a/kernel/sched/sched.h b/kernel/sched/sched.h
index cd2f1e1..76f6839 100644
--- a/kernel/sched/sched.h
+++ b/kernel/sched/sched.h
@@ -237,6 +237,7 @@ struct cfs_rq {
  struct list_head leaf_cfs_rq_list;
```

```
  struct task_group *tg; /* group that "owns" this runqueue */

+ u64 nr_switches;
 #ifdef CONFIG_SMP
  /*
   *   h_load = weight * f(tg)
@@ -307,6 +308,8 @@ struct rt_rq {
  struct rq *rq;
  struct list_head leaf_rt_rq_list;
  struct task_group *tg;
+
+ u64 rt_nr_switches;
 #endif
 };

--
1.7.10.2
```