Subject: [RFC 7/7] Global optimization
Posted by Glauber Costa on Fri, 30 Mar 2012 08:04:45 GMT
View Forum Message <> Reply to Message

When we are close to the limit, doing percpu_counter_add and its
equivalent tests is a waste of time.

This patch introduce a "global" state flag to the res_counter.
When we are close to the limit, this flag is set and we skip directly
to the locked part. The flag is unset when we are far enough away from
the limit.

In this mode, we function very much like the original resource counter

The main difference right now is that we still scan all the cpus.
This should however be very easy to avoid, with a flusher function
that empties the per-cpu areas, and then updating usage_pcp directly.

This should be doable because once we get the global flag, we know
no one else would be adding to the percpu areas any longer.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/res_counter.h | 1 +
 kernel/res_counter.c | 18 ++++++++++++++++++
 2 files changed, 19 insertions(+), 0 deletions(-)

```
diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
index 3527827..a8e4646 100644
--- a/include/linux/res_counter.h
+++ b/include/linux/res_counter.h
@@ -30,6 +30,7 @@ struct res_counter {
   * the limit that usage cannot exceed
   */
  unsigned long long limit;
+ bool global;
  /*
   * the limit that usage can be exceed
   */
diff --git a/kernel/res_counter.c b/kernel/res_counter.c
index 7b05208..859a27d 100644
--- a/kernel/res_counter.c
+++ b/kernel/res_counter.c
@@ -29,6 +29,8 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
  u64 usage;

  rcu_read_lock();
+ if (c->global)
```

```
+  goto global;

  if (val < 0) {
   percpu_counter_add(&c->usage_pcp, val);
@@ -45,9 +47,25 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
   return 0;
  }

+global:
  rcu_read_unlock();

  raw_spin_lock(&c->usage_pcp.lock);
+ usage = __percpu_counter_sum_locked(&c->usage_pcp);
+
+ /* everyone that could update global is under lock
+  * reader could miss a transition, but that is not a problem,
+  * since we are always using percpu_counter_sum anyway
+  */
+
+ if (!c->global && val > 0 && usage + val >
+     (c->limit + num_online_cpus() * percpu_counter_batch))
+  c->global = true;
+
+ if (c->global && val < 0 && usage + val <
+     (c->limit + num_online_cpus() * percpu_counter_batch))
+  c->global = false;
+

  usage = __percpu_counter_sum_locked(&c->usage_pcp);

--
1.7.4.1
```