Subject: [RFC 5/7] use percpu_counters for res_counter usage
Posted by Glauber Costa on Fri, 30 Mar 2012 08:04:43 GMT
View Forum Message <> Reply to Message

This is the bulk of the proposal.
Updates to the res_counter are done to the percpu area, if we are
inside what we can call the "safe zone".

The safe zone is whenever we are far enough from the limit to be
sure this update won't touch it. It is bigger the bigger the system
is, since it grows with the number of cpus.

However, for unlimited scenarios, this will always be the case.
In those situations we are sure to never be close to the limit simply
because the limit is high enough.

Small consumers will also be safe. This includes workloads that
pin and unpin memory often, but never grow the total size of memory
by too much.

The memory reported (reads of RES_USAGE) in this way is actually
more precise than we currently have (Actually would be, if we
would disable the memcg caches): I am using percpu_counter_sum(),
meaning the cpu areas will be scanned and accumulated.

percpu_counter_read() can also be used for reading RES_USAGE.
We could then be off by a factor of batch_size * #cpus. I consider
this to be not worse than the current situation with the memcg caches.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/res_counter.h | 15 ++++++----
 kernel/res_counter.c | 61 ++++++++++++++++++++++++++++++++++++++-------
 2 files changed, 60 insertions(+), 16 deletions(-)

diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
index 53b271c..8c1c20e 100644
--- a/include/linux/res_counter.h
+++ b/include/linux/res_counter.h
@@ -25,7 +25,6 @@ struct res_counter {
  /*
   * the current resource consumption level
   */
- unsigned long long usage;
  struct percpu_counter usage_pcp;
  /*
   * the maximal value of the usage from the counter creation
@@ -138,10 +137,12 @@ static inline unsigned long long res_counter_margin(struct res_counter

```
*cnt)
 {
  unsigned long long margin;
  unsigned long flags;
+ u64 usage;

  raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
- if (cnt->limit > cnt->usage)
-  margin = cnt->limit - cnt->usage;
+ usage = __percpu_counter_sum_locked(&cnt->usage_pcp);
+ if (cnt->limit > usage)
+  margin = cnt->limit - usage;
  else
   margin = 0;
  raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
@@ -160,12 +161,14 @@ res_counter_soft_limit_excess(struct res_counter *cnt)
 {
  unsigned long long excess;
  unsigned long flags;
+ u64 usage;

  raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
- if (cnt->usage <= cnt->soft_limit)
+ usage = __percpu_counter_sum_locked(&cnt->usage_pcp);
+ if (usage <= cnt->soft_limit)
   excess = 0;
  else
-  excess = cnt->usage - cnt->soft_limit;
+  excess = usage - cnt->soft_limit;
  raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
  return excess;
 }
@@ -175,7 +178,7 @@ static inline void res_counter_reset_max(struct res_counter *cnt)
  unsigned long flags;

  raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
- cnt->max_usage = cnt->usage;
+ cnt->max_usage = __percpu_counter_sum_locked(&cnt->usage_pcp);
  raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
 }

diff --git a/kernel/res_counter.c b/kernel/res_counter.c
index 052efaf..8a99943 100644
--- a/kernel/res_counter.c
+++ b/kernel/res_counter.c
@@ -28,9 +28,28 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
  int ret = 0;
  u64 usage;
```

```
+ rcu_read_lock();
+
+ if (val < 0) {
+  percpu_counter_add(&c->usage_pcp, val);
+  rcu_read_unlock();
+  return 0;
+ }
+
+ usage = percpu_counter_read(&c->usage_pcp);
+
+ if (percpu_counter_read(&c->usage_pcp) + val <
+     (c->limit + num_online_cpus() * percpu_counter_batch)) {
+  percpu_counter_add(&c->usage_pcp, val);
+  rcu_read_unlock();
+  return 0;
+ }
+
+ rcu_read_unlock();
+
  raw_spin_lock(&c->usage_pcp.lock);

- usage = c->usage;
+ usage = __percpu_counter_sum_locked(&c->usage_pcp);

  if (usage + val > c->limit) {
   c->failcnt++;
@@ -39,9 +58,9 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
    goto out;
  }

- usage += val;

- c->usage = usage;
+ c->usage_pcp.count += val;
+
  if (usage > c->max_usage)
   c->max_usage = usage;

@@ -115,14 +134,28 @@ int res_counter_set_limit(struct res_counter *cnt,
      unsigned long long limit)
 {
  unsigned long flags;
- int ret = -EBUSY;
+ int ret = 0;
+ u64 usage;
+ bool allowed;
```

```
+ /*
+  * This is to prevent conflicts with people reading
+  * from the pcp counters
+  */
+ synchronize_rcu();
  raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
- if (cnt->usage <= limit) {
-  cnt->limit = limit;
-  ret = 0;
+
+ usage = __percpu_counter_sum_locked(&cnt->usage_pcp);
+ if (usage >= limit) {
+  allowed = false;
+  ret = -EBUSY;
+  goto out;
  }
+
+ cnt->limit = limit;
+out:
  raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
+
  return ret;
 }

@@ -130,8 +163,6 @@ static inline unsigned long long *
 res_counter_member(struct res_counter *counter, int member)
 {
  switch (member) {
- case RES_USAGE:
-  return &counter->usage;
  case RES_MAX_USAGE:
   return &counter->max_usage;
  case RES_LIMIT:
@@ -153,7 +184,11 @@ u64 res_counter_read_u64(struct res_counter *counter, int member)
  u64 ret;

  raw_spin_lock_irqsave(&counter->usage_pcp.lock, flags);
- ret = *res_counter_member(counter, member);
+ if (member == RES_USAGE) {
+  synchronize_rcu();
+  ret = __percpu_counter_sum_locked(&counter->usage_pcp);
+ } else
+  ret = *res_counter_member(counter, member);
  raw_spin_unlock_irqrestore(&counter->usage_pcp.lock, flags);

  return ret;
@@ -161,6 +196,12 @@ u64 res_counter_read_u64(struct res_counter *counter, int member)
 #else
```

```
 u64 res_counter_read_u64(struct res_counter *counter, int member)
 {
+ if (member == RES_USAGE) {
+  u64 ret;
+  synchronize_rcu();
+  ret = percpu_counter_sum(&counter->usage_pcp);
+  return ret;
+ }
  return *res_counter_member(counter, member);
 }
 #endif
--
1.7.4.1
```