Subject: [PATCH 09/11] memcg: propagate kmem limiting information to children
Posted by Glauber Costa on Mon, 25 Jun 2012 14:15:26 GMT
View Forum Message <> Reply to Message

The current memcg slab cache management fails to present satisfatory hierarchical
behavior in the following scenario:

-> /cgroups/memory/A/B/C

* kmem limit set at A
* A and B empty taskwise
* bash in C does find /

Because kmem_accounted is a boolean that was not set for C, no accounting
would be done. This is, however, not what we expect.

The basic idea, is that when a cgroup is limited, we walk the tree
upwards (something Kame and I already thought about doing for other purposes),
and make sure that we store the information about the parent being limited in
kmem_accounted (that is turned into a bitmap: two booleans would not be space
efficient). The code for that is taken from sched/core.c. My reasons for not
putting it into a common place is to dodge the type issues that would arise
from a common implementation between memcg and the scheduler - but I think
that it should ultimately happen, so if you want me to do it now, let me
know.

We do the reverse operation when a formerly limited cgroup becomes unlimited.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: Michal Hocko <mhocko@suse.cz>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Suleiman Souhlal <suleiman@google.com>

```
---
 mm/memcontrol.c |   86 +++++++++++++++++++++++++++++++++++++++++++++----------
 1 file changed, 71 insertions(+), 15 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index fe5388e..a6a440b 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -287,7 +287,11 @@ struct mem_cgroup {
 	 * Should the accounting and control be hierarchical, per subtree?
 	 */
 	bool use_hierarchy;
-	bool kmem_accounted;
```

```
+ /*
+  * bit0: accounted by this cgroup
+  * bit1: accounted by a parent.
+  */
+ volatile unsigned long kmem_accounted;

  bool  oom_lock;
  atomic_t under_oom;
@@ -340,6 +344,9 @@ struct mem_cgroup {
 #endif
 };

+#define KMEM_ACCOUNTED_THIS 0
+#define KMEM_ACCOUNTED_PARENT 1
+
 /* Stuffs for move charges at task migration. */
 /*
  * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
@@ -589,7 +596,7 @@ EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);

 static void disarm_kmem_keys(struct mem_cgroup *memcg)
 {
- if (memcg->kmem_accounted)
+ if (test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted))
   static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
 }
 #else
@@ -4027,6 +4034,66 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
  len = scnprintf(str, sizeof(str), "%llu\n", (unsigned long long)val);
  return simple_read_from_buffer(buf, nbytes, ppos, str, len);
 }
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static void mem_cgroup_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
+{
+ struct mem_cgroup *iter;
+
+ mutex_lock(&set_limit_mutex);
+ if (!test_and_set_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted) &&
+  val != RESOURCE_MAX) {
+
+  /*
+   * Once enabled, can't be disabled. We could in theory
+   * disable it if we haven't yet created any caches, or
+   * if we can shrink them all to death.
+   *
+   * But it is not worth the trouble
```

```
+   */
+  static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
+
+  if (!memcg->use_hierarchy)
+   goto out;
+
+  for_each_mem_cgroup_tree(iter, memcg) {
+   if (iter == memcg)
+    continue;
+   set_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
+  }
+
+ } else if (test_and_clear_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted)
+  && val == RESOURCE_MAX) {
+
+  if (!memcg->use_hierarchy)
+   goto out;
+
+  for_each_mem_cgroup_tree(iter, memcg) {
+   struct mem_cgroup *parent;
+   if (iter == memcg)
+    continue;
+   /*
+    * We should only have our parent bit cleared if none of
+    * ouri parents are accounted. The transversal order of
+    * our iter function forces us to always look at the
+    * parents.
+    */
+   parent = parent_mem_cgroup(iter);
+   while (parent && (parent != memcg)) {
+    if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
+     goto noclear;
+
+    parent = parent_mem_cgroup(parent);
+   }
+   clear_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
+noclear:
+   continue;
+  }
+ }
+out:
+ mutex_unlock(&set_limit_mutex);
+}
+#endif
 /*
  * The user of this function is...
  * RES_LIMIT.
@@ -4064,19 +4131,8 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
```

```
    ret = res_counter_set_limit(&memcg->kmem, val);
    if (ret)
     break;
-   /*
-    * Once enabled, can't be disabled. We could in theory
-    * disable it if we haven't yet created any caches, or
-    * if we can shrink them all to death.
-    *
-    * But it is not worth the trouble
-    */
-   mutex_lock(&set_limit_mutex);
-   if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
-    static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
-    memcg->kmem_accounted = true;
-   }
-   mutex_unlock(&set_limit_mutex);
+   mem_cgroup_update_kmem_limit(memcg, val);
+   break;
   }
 #endif
   else
--
1.7.10.2
```