Subject: [PATCH v2 09/11] memcg: propagate kmem limiting information to children
Posted by Glauber Costa on Thu, 09 Aug 2012 13:01:17 GMT
View Forum Message <> Reply to Message

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

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

* kmem limit set at A,
* A and B have no tasks,
* span a new task in in C.

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 | 88 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++------
 1 file changed, 79 insertions(+), 9 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 3216292..3d30b79 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -295,7 +295,8 @@ struct mem_cgroup {
   * Should the accounting and control be hierarchical, per subtree?
   */
  bool use_hierarchy;
```

```
- bool kmem_accounted;
+
+ unsigned long kmem_accounted; /* See KMEM_ACCOUNTED_*, below */

  bool  oom_lock;
  atomic_t under_oom;
@@ -348,6 +349,38 @@ struct mem_cgroup {
 #endif
 };

+enum {
+ KMEM_ACCOUNTED_THIS, /* accounted by this cgroup itself */
+ KMEM_ACCOUNTED_PARENT, /* accounted by any of its parents. */
+};
+
+#ifdef CONFIG_MEMCG_KMEM
+static bool memcg_kmem_account(struct mem_cgroup *memcg)
+{
+ return !test_and_set_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted);
+}
+
+static bool memcg_kmem_clear_account(struct mem_cgroup *memcg)
+{
+ return test_and_clear_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted);
+}
+
+static bool memcg_kmem_is_accounted(struct mem_cgroup *memcg)
+{
+ return test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted);
+}
+
+static void memcg_kmem_account_parent(struct mem_cgroup *memcg)
+{
+ set_bit(KMEM_ACCOUNTED_PARENT, &memcg->kmem_accounted);
+}
+
+static void memcg_kmem_clear_account_parent(struct mem_cgroup *memcg)
+{
+ clear_bit(KMEM_ACCOUNTED_PARENT, &memcg->kmem_accounted);
+}
+#endif /* CONFIG_MEMCG_KMEM */
+
 /* Stuffs for move charges at task migration. */
 /*
  * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
@@ -614,7 +647,7 @@ EXPORT_SYMBOL(__memcg_kmem_free_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(&memcg_kmem_enabled_key);
 }
 #else
@@ -4171,17 +4204,54 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
 static void memcg_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
 {
 #ifdef CONFIG_MEMCG_KMEM
- /*
-  * 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.
-  */
+ struct mem_cgroup *iter;
+
   mutex_lock(&set_limit_mutex);
- if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
+ if ((val != RESOURCE_MAX) && memcg_kmem_account(memcg)) {
+
+  /*
+   * 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(&memcg_kmem_enabled_key);
-  memcg->kmem_accounted = true;
+
+  if (!memcg->use_hierarchy)
+   goto out;
+
+  for_each_mem_cgroup_tree(iter, memcg) {
+   if (iter == memcg)
+    continue;
+   memcg_kmem_account_parent(iter);
+  }
+ } else if ((val == RESOURCE_MAX) && memcg_kmem_clear_account(memcg)) {
+
+  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 our parents are accounted. The transversal order
+    * of our iter function forces us to always look at the
+    * parents.
+    */
+   parent = parent_mem_cgroup(iter);
+   for (; parent != memcg; parent = parent_mem_cgroup(iter))
+    if (memcg_kmem_is_accounted(parent))
+     goto noclear;
+   memcg_kmem_clear_account_parent(iter);
+noclear:
+   continue;
+  }
  }
+out:
  mutex_unlock(&set_limit_mutex);
+
 #endif
 }

--
1.7.11.2
```