Subject: [PATCH v4 09/14] memcg: kmem accounting lifecycle management
Posted by Glauber Costa on Mon, 08 Oct 2012 10:06:15 GMT
View Forum Message <> Reply to Message

Because kmem charges can outlive the cgroup, we need to make sure that
we won't free the memcg structure while charges are still in flight.
For reviewing simplicity, the charge functions will issue
mem_cgroup_get() at every charge, and mem_cgroup_put() at every
uncharge.

This can get expensive, however, and we can do better. mem_cgroup_get()
only really needs to be issued once: when the first limit is set. In the
same spirit, we only need to issue mem_cgroup_put() when the last charge
is gone.

We'll need an extra bit in kmem_accounted for that: KMEM_ACCOUNTED_DEAD.
it will be set when the cgroup dies, if there are charges in the group.
If there aren't, we can proceed right away.

Our uncharge function will have to test that bit every time the charges
drop to 0. Because that is not the likely output of
res_counter_uncharge, this should not impose a big hit on us: it is
certainly much better than a reference count decrease at every
operation.

[v3: merged all lifecycle related patches in one]

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: Michal Hocko <mhocko@suse.cz>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Suleiman Souhlal <suleiman@google.com>
---
 mm/memcontrol.c | 56 ++-------
 1 file changed, 49 insertions(+), 7 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index dda54f0..634c7b5 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -344,6 +344,7 @@ struct mem_cgroup {
 /* internal only representation about the status of kmem accounting. */
 enum {
  KMEM_ACCOUNTED_ACTIVE = 0, /* accounted by this cgroup itself */
+ KMEM_ACCOUNTED_DEAD, /* dead memcg, pending kmem charges */
 };

```
 /* first bit */
@@ -354,6 +355,22 @@ static void memcg_kmem_set_active(struct mem_cgroup *memcg)
 {
  set_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted);
 }
+
+static bool memcg_kmem_is_active(struct mem_cgroup *memcg)
+{
+ return test_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted);
+}
+
+static void memcg_kmem_mark_dead(struct mem_cgroup *memcg)
+{
+ if (test_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted))
+  set_bit(KMEM_ACCOUNTED_DEAD, &memcg->kmem_accounted);
+}
+
+static bool memcg_kmem_dead(struct mem_cgroup *memcg)
+{
+ return test_and_clear_bit(KMEM_ACCOUNTED_DEAD, &memcg->kmem_accounted);
+}
 #endif

 /* Stuffs for move charges at task migration. */
@@ -2690,10 +2707,16 @@ static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t
gfp, u64 size)

 static void memcg_uncharge_kmem(struct mem_cgroup *memcg, u64 size)
 {
- res_counter_uncharge(&memcg->kmem, size);
  res_counter_uncharge(&memcg->res, size);
  if (do_swap_account)
   res_counter_uncharge(&memcg->memsw, size);
+
+ /* Not down to 0 */
+ if (res_counter_uncharge(&memcg->kmem, size))
+  return;
+
+ if (memcg_kmem_dead(memcg))
+  mem_cgroup_put(memcg);
 }

 /*
@@ -2732,13 +2755,9 @@ __memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup
**_memcg, int order)
   return true;
  }
```

```
- mem_cgroup_get(memcg);
-
  ret = memcg_charge_kmem(memcg, gfp, PAGE_SIZE << order);
  if (!ret)
   *_memcg = memcg;
- else
-  mem_cgroup_put(memcg);

  css_put(&memcg->css);
  return (ret == 0);
@@ -2754,7 +2773,6 @@ void __memcg_kmem_commit_charge(struct page *page, struct
mem_cgroup *memcg,
  /* The page allocation failed. Revert */
  if (!page) {
   memcg_uncharge_kmem(memcg, PAGE_SIZE << order);
-  mem_cgroup_put(memcg);
   return;
  }

@@ -2795,7 +2813,6 @@ void __memcg_kmem_uncharge_page(struct page *page, int order)

  VM_BUG_ON(mem_cgroup_is_root(memcg));
  memcg_uncharge_kmem(memcg, PAGE_SIZE << order);
- mem_cgroup_put(memcg);
 }
 #endif /* CONFIG_MEMCG_KMEM */

@@ -4180,6 +4197,13 @@ static int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
    goto out;

   memcg_kmem_set_active(memcg);
+  /*
+   * kmem charges can outlive the cgroup. In the case of slab
+   * pages, for instance, a page contain objects from various
+   * processes, so it is unfeasible to migrate them away. We
+   * need to reference count the memcg because of that.
+   */
+  mem_cgroup_get(memcg);
  } else
   ret = res_counter_set_limit(&memcg->kmem, val);
 out:
@@ -4193,6 +4217,10 @@ static void memcg_propagate_kmem(struct mem_cgroup *memcg,
      struct mem_cgroup *parent)
 {
  memcg->kmem_accounted = parent->kmem_accounted;
+#ifdef CONFIG_MEMCG_KMEM
+ if (memcg_kmem_is_active(memcg))
```

```
+  mem_cgroup_get(memcg);
+#endif
 }

 /*
@@ -4876,6 +4904,20 @@ static int memcg_init_kmem(struct mem_cgroup *memcg, struct
cgroup_subsys *ss)
 static void kmem_cgroup_destroy(struct mem_cgroup *memcg)
 {
  mem_cgroup_sockets_destroy(memcg);
+
+ memcg_kmem_mark_dead(memcg);
+
+ if (res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0)
+  return;
+
+ /*
+  * Charges already down to 0, undo mem_cgroup_get() done in the charge
+  * path here, being careful not to race with memcg_uncharge_kmem: it is
+  * possible that the charges went down to 0 between mark_dead and the
+  * res_counter read, so in that case, we don't need the put
+  */
+ if (memcg_kmem_dead(memcg))
+  mem_cgroup_put(memcg);
 }
 #else
 static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
--
1.7.11.4
```