Subject: [PATCH v3 26/28] slub: track all children of a kmem cache
Posted by Glauber Costa on Fri, 25 May 2012 13:03:46 GMT
View Forum Message <> Reply to Message

When we destroy a cache (like for instance, if we're unloading a module)
we need to go through the list of memcg caches and destroy them as well.

The caches are expected to be empty by themselves, so nothing is changed
here. All previous guarantees are kept and no new guarantees are given.

So given all memcg caches are expected to be empty - even though they are
likely to be hanging around in the system, we just need to scan a list of
sibling caches, and destroy each one of them.

This is very similar to the work done by Suleiman for the slab.

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/slub.c | 64 ++--------
 1 files changed, 55 insertions(+), 9 deletions(-)

diff --git a/mm/slub.c b/mm/slub.c
index 4c29e5f..8151353 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -3254,6 +3254,54 @@ static inline int kmem_cache_close(struct kmem_cache *s)
  return 0;
 }

+static void kmem_cache_destroy_unlocked(struct kmem_cache *s)
+{
+ mem_cgroup_release_cache(s);
+ if (kmem_cache_close(s)) {
+ printk(KERN_ERR
+   "SLUB %s: %s called for cache that still has objects.\n",
+   s->name, __func__);
+ dump_stack();
+ }
+
+ if (s->flags & SLAB_DESTROY_BY_RCU)
+ rcu_barrier();
+ sysfs_slab_remove(s);

```
+}
+
+static void kmem_cache_destroy_memcg_children(struct kmem_cache *s)
+{
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ struct mem_cgroup_cache_params *p, *tmp, *this;
+ struct kmem_cache *c;
+ int id = s->memcg_params.id;
+
+ /* Not a parent cache */
+ if (id == -1)
+  return ;
+
+ this = &s->memcg_params;
+ mem_cgroup_flush_cache_create_queue();
+ list_for_each_entry_safe(p, tmp, &this->sibling_list, sibling_list) {
+  c = container_of(p, struct kmem_cache, memcg_params);
+  /* We never added the main cache to the sibling list */
+  if (WARN_ON(c == s))
+   continue;
+
+  c->refcount--;
+  if (c->refcount)
+   continue;
+
+  list_del(&c->list);
+  list_del(&c->memcg_params.sibling_list);
+  s->refcount--; /* parent reference */
+  up_write(&slub_lock);
+  mem_cgroup_remove_child_kmem_cache(c, id);
+  kmem_cache_destroy_unlocked(c);
+  down_write(&slub_lock);
+ }
+#endif
+}
 /*
  * Close a cache and release the kmem_cache structure
  * (must be used for caches created using kmem_cache_create)
@@ -3261,19 +3309,15 @@ static inline int kmem_cache_close(struct kmem_cache *s)
 void kmem_cache_destroy(struct kmem_cache *s)
 {
  down_write(&slub_lock);
+ kmem_cache_destroy_memcg_children(s);
  s->refcount--;
  if (!s->refcount) {
   list_del(&s->list);
-  mem_cgroup_release_cache(s);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
```

```
+  list_del(&s->memcg_params.sibling_list);
+#endif
   up_write(&slub_lock);
-  if (kmem_cache_close(s)) {
-   printk(KERN_ERR "SLUB %s: %s called for cache that "
-    "still has objects.\n", s->name, __func__);
-   dump_stack();
-  }
-  if (s->flags & SLAB_DESTROY_BY_RCU)
-   rcu_barrier();
-  sysfs_slab_remove(s);
+  kmem_cache_destroy_unlocked(s);
  } else
   up_write(&slub_lock);
 }
@@ -4102,6 +4146,8 @@ struct kmem_cache *kmem_cache_dup(struct mem_cgroup *memcg,
   */
  if (new) {
   down_write(&slub_lock);
+  list_add(&new->memcg_params.sibling_list,
+    &s->memcg_params.sibling_list);
   s->refcount++;
   up_write(&slub_lock);
   slab_set_parent(new, s);
--
1.7.7.6
```