Subject: [PATCH v2 23/29] memcg: destroy memcg caches
Posted by Glauber Costa on Fri, 11 May 2012 17:44:25 GMT
View Forum Message <> Reply to Message

This patch implements destruction of memcg caches. Right now,
only caches where our reference counter is the last remaining are
deleted. If there are any other reference counters around, we just
leave the caches lying around until they go away.

When that happen, a destruction function is called from the cache
code. Caches are only destroyed in process context, so we queue them
up for later processing in the general case.

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>
---
 include/linux/memcontrol.h | 2 +
 include/linux/slab.h | 1 +
 mm/memcontrol.c | 91 ++-
 mm/slab.c | 5 +-
 mm/slub.c | 7 ++-
 5 files changed, 101 insertions(+), 5 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 4000798..3e03f26 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -463,6 +463,8 @@ __mem_cgroup_get_kmem_cache(struct kmem_cache *cachep, gfp_t gfp);

 extern struct static_key mem_cgroup_kmem_enabled_key;
 #define mem_cgroup_kmem_on static_key_false(&mem_cgroup_kmem_enabled_key)
+
+void mem_cgroup_destroy_cache(struct kmem_cache *cachep);
 #else
 static inline void mem_cgroup_register_cache(struct mem_cgroup *memcg,
          struct kmem_cache *s)
diff --git a/include/linux/slab.h b/include/linux/slab.h
index e73ef71..a03a4f2 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -164,6 +164,7 @@ struct mem_cgroup_cache_params {
  size_t orig_align;

```
 #endif
+ struct list_head destroyed_list; /* Used when deleting memcg cache */
 };
 #endif

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index ad60648..1d1a307 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -476,6 +476,11 @@ static void disarm_static_keys(struct mem_cgroup *memcg)
 {
  if (memcg->kmem_accounted)
   static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
+ /*
+  * This check can't live in kmem destruction function,
+  * since the charges will outlive the cgroup
+  */
+ BUG_ON(res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0);
 }

 #ifdef CONFIG_INET
@@ -540,6 +545,8 @@ void mem_cgroup_register_cache(struct mem_cgroup *memcg,
  if (!memcg)
   id = ida_simple_get(&cache_types, 0, MAX_KMEM_CACHE_TYPES,
         GFP_KERNEL);
+ else
+  INIT_LIST_HEAD(&cachep->memcg_params.destroyed_list);
  cachep->memcg_params.id = id;
 }

@@ -592,6 +599,53 @@ struct create_work {
 /* Use a single spinlock for destruction and creation, not a frequent op */
 static DEFINE_SPINLOCK(cache_queue_lock);
 static LIST_HEAD(create_queue);
+static LIST_HEAD(destroyed_caches);
+
+static void kmem_cache_destroy_work_func(struct work_struct *w)
+{
+ struct kmem_cache *cachep;
+ struct mem_cgroup_cache_params *p, *tmp;
+ unsigned long flags;
+ LIST_HEAD(del_unlocked);
+
+ spin_lock_irqsave(&cache_queue_lock, flags);
+ list_for_each_entry_safe(p, tmp, &destroyed_caches, destroyed_list) {
+  cachep = container_of(p, struct kmem_cache, memcg_params);
+  list_move(&cachep->memcg_params.destroyed_list, &del_unlocked);
```

```
+ }
+ spin_unlock_irqrestore(&cache_queue_lock, flags);
+
+ list_for_each_entry_safe(p, tmp, &del_unlocked, destroyed_list) {
+  cachep = container_of(p, struct kmem_cache, memcg_params);
+  list_del(&cachep->memcg_params.destroyed_list);
+  if (!atomic_read(&cachep->memcg_params.refcnt)) {
+   mem_cgroup_put(cachep->memcg_params.memcg);
+   kmem_cache_destroy(cachep);
+  }
+ }
+}
+static DECLARE_WORK(kmem_cache_destroy_work, kmem_cache_destroy_work_func);
+
+static void __mem_cgroup_destroy_cache(struct kmem_cache *cachep)
+{
+ BUG_ON(cachep->memcg_params.id != -1);
+ list_add(&cachep->memcg_params.destroyed_list, &destroyed_caches);
+}
+
+void mem_cgroup_destroy_cache(struct kmem_cache *cachep)
+{
+ unsigned long flags;
+
+ /*
+  * We have to defer the actual destroying to a workqueue, because
+  * we might currently be in a context that cannot sleep.
+  */
+ spin_lock_irqsave(&cache_queue_lock, flags);
+ __mem_cgroup_destroy_cache(cachep);
+ spin_unlock_irqrestore(&cache_queue_lock, flags);
+
+ schedule_work(&kmem_cache_destroy_work);
+}

 /*
  * Flush the queue of kmem_caches to create, because we're creating a cgroup.
@@ -613,6 +667,33 @@ void mem_cgroup_flush_cache_create_queue(void)
  spin_unlock_irqrestore(&cache_queue_lock, flags);
 }

+static void mem_cgroup_destroy_all_caches(struct mem_cgroup *memcg)
+{
+ struct kmem_cache *cachep;
+ unsigned long flags;
+ int i;
+
+ /*
```

```
+  * pre_destroy() gets called with no tasks in the cgroup.
+  * this means that after flushing the create queue, no more caches
+  * will appear
+  */
+ mem_cgroup_flush_cache_create_queue();
+
+ spin_lock_irqsave(&cache_queue_lock, flags);
+ for (i = 0; i < MAX_KMEM_CACHE_TYPES; i++) {
+  cachep = memcg->slabs[i];
+  if (!cachep)
+   continue;
+
+  if (atomic_dec_and_test(&cachep->memcg_params.refcnt))
+   __mem_cgroup_destroy_cache(cachep);
+ }
+ spin_unlock_irqrestore(&cache_queue_lock, flags);
+
+ schedule_work(&kmem_cache_destroy_work);
+}
+
 static void memcg_create_cache_work_func(struct work_struct *w)
 {
  struct create_work *cw, *tmp;
@@ -854,6 +935,10 @@ static void memcg_slab_init(struct mem_cgroup *memcg)
 static inline void disarm_static_keys(struct mem_cgroup *memcg)
 {
 }
+
+static inline void mem_cgroup_destroy_all_caches(struct mem_cgroup *memcg)
+{
+}
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

 static void drain_all_stock_async(struct mem_cgroup *memcg);
@@ -4133,6 +4218,7 @@ static int mem_cgroup_force_empty(struct mem_cgroup *memcg, bool
free_all)
  int node, zid, shrink;
  int nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
  struct cgroup *cgrp = memcg->css.cgroup;
+ u64 usage;

  css_get(&memcg->css);

@@ -4172,8 +4258,10 @@ move_account:
   if (ret == -ENOMEM)
    goto try_to_free;
   cond_resched();
+  usage = res_counter_read_u64(&memcg->res, RES_USAGE) -
```

```
+   res_counter_read_u64(&memcg->kmem, RES_USAGE);
  /* "ret" should also be checked to ensure all lists are empty. */
- } while (res_counter_read_u64(&memcg->res, RES_USAGE) > 0 || ret);
+ } while (usage > 0 || ret);
 out:
  css_put(&memcg->css);
  return ret;
@@ -5518,6 +5606,7 @@ static int mem_cgroup_pre_destroy(struct cgroup *cont)
 {
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);

+ mem_cgroup_destroy_all_caches(memcg);
  return mem_cgroup_force_empty(memcg, false);
 }

diff --git a/mm/slab.c b/mm/slab.c
index 7022f86..a6fd82e 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1861,8 +1861,9 @@ static void *kmem_getpages(struct kmem_cache *cachep, gfp_t flags,
int nodeid)
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 void kmem_cache_drop_ref(struct kmem_cache *cachep)
 {
- if (cachep->memcg_params.id == -1)
-  atomic_dec(&cachep->memcg_params.refcnt);
+ if (cachep->memcg_params.id == -1 &&
+     unlikely(atomic_dec_and_test(&cachep->memcg_params.refcnt)))
+  mem_cgroup_destroy_cache(cachep);
 }
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

diff --git a/mm/slub.c b/mm/slub.c
index c70db56..02d8f5e 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1297,8 +1297,11 @@ static void kmem_cache_inc_ref(struct kmem_cache *s)
 }
 static void kmem_cache_drop_ref(struct kmem_cache *s)
 {
- if (s->memcg_params.memcg)
-  atomic_dec(&s->memcg_params.refcnt);
+ if (!s->memcg_params.memcg)
+  return;
+
+ if (unlikely(atomic_dec_and_test(&s->memcg_params.refcnt)))
+  mem_cgroup_destroy_cache(s);
 }
```

#else
static inline void kmem_cache_inc_ref(struct kmem_cache *s)
--
1.7.7.6