Subject: [PATCH v5 12/18] memcg: destroy memcg caches
Posted by Glauber Costa on Fri, 19 Oct 2012 14:20:36 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.

[ v5: removed cachep backpointer ]

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>
CC: Tejun Heo <tj@kernel.org>
---
 include/linux/memcontrol.h | 2 ++
 include/linux/slab.h | 7 +++++++
 mm/memcontrol.c | 51 ++++++

```
+++ b/include/linux/slab.h
@@ -198,6 +198,10 @@ unsigned int kmem_cache_size(struct kmem_cache *);
  *
  * @memcg: pointer to the memcg this cache belongs to
  * @root_cache: pointer to the global, root cache, this cache was derived from
+ * @dead: set to true after the memcg dies; the cache may still be around.
+ * @nr_pages: number of pages that belongs to this cache.
+ * @destroy: worker to be called whenever we are ready, or believe we may be
+ *           ready, to destroy this cache.
  */
 struct memcg_cache_params {
  bool is_root_cache;
@@ -206,6 +210,9 @@ struct memcg_cache_params {
    struct {
     struct mem_cgroup *memcg;
     struct kmem_cache *root_cache;
+    bool dead;
+    atomic_t nr_pages;
+    struct work_struct destroy;
    };
   };
 };
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 354cdf0..0359b3a 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -2950,6 +2950,33 @@ out:
  kfree(s->memcg_params);
 }

+static void kmem_cache_destroy_work_func(struct work_struct *w)
+{
+ struct kmem_cache *cachep;
+ struct memcg_cache_params *p;
+
+ p = container_of(w, struct memcg_cache_params, destroy);
+
+ VM_BUG_ON(p->is_root_cache);
+ cachep = p->root_cache;
+ cachep = cachep->memcg_params->memcg_caches[memcg_css_id(p->memcg)];
+
+ if (!atomic_read(&cachep->memcg_params->nr_pages))
+  kmem_cache_destroy(cachep);
+}
+
+void mem_cgroup_destroy_cache(struct kmem_cache *cachep)
+{
+ if (!cachep->memcg_params->dead)
```

```
+  return;
+
+ /*
+  * We have to defer the actual destroying to a workqueue, because
+  * we might currently be in a context that cannot sleep.
+  */
+ schedule_work(&cachep->memcg_params->destroy);
+}
+
 /*
  * During the creation a new cache, we need to disable our accounting mechanism
  * altogether. This is true even if we are not creating, but rather just
@@ -3062,6 +3089,7 @@ static struct kmem_cache *memcg_create_kmem_cache(struct
mem_cgroup *memcg,
  wmb(); /* the readers won't lock, make sure everybody sees it */
  new_cachep->memcg_params->memcg = memcg;
  new_cachep->memcg_params->root_cache = cachep;
+ atomic_set(&new_cachep->memcg_params->nr_pages , 0);
 out:
  mutex_unlock(&memcg_cache_mutex);
  return new_cachep;
@@ -3073,6 +3101,24 @@ struct create_work {
  struct work_struct work;
 };

+static void mem_cgroup_destroy_all_caches(struct mem_cgroup *memcg)
+{
+ struct kmem_cache *cachep;
+
+ if (!memcg_kmem_is_active(memcg))
+  return;
+
+ mutex_lock(&memcg->slab_caches_mutex);
+ list_for_each_entry(cachep, &memcg->memcg_slab_caches, list) {
+
+  cachep->memcg_params->dead = true;
+  INIT_WORK(&cachep->memcg_params->destroy,
+      kmem_cache_destroy_work_func);
+  schedule_work(&cachep->memcg_params->destroy);
+ }
+ mutex_unlock(&memcg->slab_caches_mutex);
+}
+
 static void memcg_create_cache_work_func(struct work_struct *w)
 {
  struct create_work *cw;
@@ -3276,6 +3322,10 @@ void __memcg_kmem_uncharge_pages(struct page *page, int order)
  VM_BUG_ON(mem_cgroup_is_root(memcg));
```

```
  memcg_uncharge_kmem(memcg, PAGE_SIZE << order);
 }
+#else
+static inline void mem_cgroup_destroy_all_caches(struct mem_cgroup *memcg)
+{
+}
 #endif /* CONFIG_MEMCG_KMEM */

 #ifdef CONFIG_TRANSPARENT_HUGEPAGE
@@ -5875,6 +5925,7 @@ 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 c6cdcc0..f29b43e4 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1911,6 +1911,7 @@ static void *kmem_getpages(struct kmem_cache *cachep, gfp_t flags,
int nodeid)
    if (page->pfmemalloc)
     SetPageSlabPfmemalloc(page + i);
  }
+ memcg_bind_pages(cachep, cachep->gfporder);

  if (kmemcheck_enabled && !(cachep->flags & SLAB_NOTRACK)) {
   kmemcheck_alloc_shadow(page, cachep->gfporder, flags, nodeid);
@@ -1947,6 +1948,8 @@ static void kmem_freepages(struct kmem_cache *cachep, void *addr)
   __ClearPageSlab(page);
   page++;
  }
+
+ memcg_release_pages(cachep, cachep->gfporder);
  if (current->reclaim_state)
   current->reclaim_state->reclaimed_slab += nr_freed;
  free_memcg_kmem_pages((unsigned long)addr, cachep->gfporder);
diff --git a/mm/slab.h b/mm/slab.h
index b9b5f1f..ab57462 100644
--- a/mm/slab.h
+++ b/mm/slab.h
@@ -1,5 +1,6 @@
 #ifndef MM_SLAB_H
 #define MM_SLAB_H
+#include <linux/memcontrol.h>
 /*
```

```
  * Internal slab definitions
  */
@@ -109,6 +110,21 @@ static inline bool cache_match_memcg(struct kmem_cache *cachep,
   (cachep->memcg_params->memcg == memcg);
 }

+static inline void memcg_bind_pages(struct kmem_cache *s, int order)
+{
+ if (!is_root_cache(s))
+  atomic_add(1 << order, &s->memcg_params->nr_pages);
+}
+
+static inline void memcg_release_pages(struct kmem_cache *s, int order)
+{
+ if (is_root_cache(s))
+  return;
+
+ if (atomic_sub_and_test((1 << order), &s->memcg_params->nr_pages))
+  mem_cgroup_destroy_cache(s);
+}
+
 static inline bool slab_equal_or_root(struct kmem_cache *s,
       struct kmem_cache *p)
 {
@@ -127,6 +143,14 @@ static inline bool cache_match_memcg(struct kmem_cache *cachep,
  return true;
 }

+static inline void memcg_bind_pages(struct kmem_cache *s, int order)
+{
+}
+
+static inline void memcg_release_pages(struct kmem_cache *s, int order)
+{
+}
+
 static inline bool slab_equal_or_root(struct kmem_cache *s,
       struct kmem_cache *p)
 {
diff --git a/mm/slub.c b/mm/slub.c
index 48b7574..b5b970b 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1344,6 +1344,7 @@ static struct page *new_slab(struct kmem_cache *s, gfp_t flags, int
node)
  void *start;
  void *last;
  void *p;
```

```
+ int order;

  BUG_ON(flags & GFP_SLAB_BUG_MASK);

@@ -1352,7 +1353,9 @@ static struct page *new_slab(struct kmem_cache *s, gfp_t flags, int
node)
  if (!page)
   goto out;

+ order = compound_order(page);
  inc_slabs_node(s, page_to_nid(page), page->objects);
+ memcg_bind_pages(s, order);
  page->slab = s;
  __SetPageSlab(page);
  if (page->pfmemalloc)
@@ -1361,7 +1364,7 @@ static struct page *new_slab(struct kmem_cache *s, gfp_t flags, int
node)
  start = page_address(page);

  if (unlikely(s->flags & SLAB_POISON))
-  memset(start, POISON_INUSE, PAGE_SIZE << compound_order(page));
+  memset(start, POISON_INUSE, PAGE_SIZE << order);

  last = start;
  for_each_object(p, s, start, page->objects) {
@@ -1402,6 +1405,8 @@ static void __free_slab(struct kmem_cache *s, struct page *page)

  __ClearPageSlabPfmemalloc(page);
  __ClearPageSlab(page);
+
+ memcg_release_pages(s, order);
  reset_page_mapcount(page);
  if (current->reclaim_state)
   current->reclaim_state->reclaimed_slab += pages;
--
1.7.11.7
```