

---

Subject: [PATCH v4 13/19] memcg: destroy memcg caches  
Posted by [Glauber Costa](#) on Fri, 12 Oct 2012 13:41:07 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>  
CC: Tejun Heo <tj@kernel.org>

---  
include/linux/memcontrol.h | 2 ++  
include/linux/slab.h | 9 ++++++++  
mm/memcontrol.c | 47 ++++++++++++++++++++++++++++++++++++++  
mm/slab.c | 3 +++  
mm/slab.h | 24 ++++++++++++++++++++++++++++++++++  
mm/slub.c | 7 +++++-  
6 files changed, 91 insertions(+), 1 deletion(-)

```
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 4c94182..9ac12cb 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -426,6 +426,8 @@ void memcg_update_array_size(int num_groups);
struct kmem_cache *
__memcg_kmem_get_cache(struct kmem_cache *cachep, gfp_t gfp);

+void mem_cgroup_destroy_cache(struct kmem_cache *cachep);
+
+/**
+ * memcg_kmem_newpage_charge: verify if a new kmem allocation is allowed.
+ * @gfp: the gfp allocation flags.
diff --git a/include/linux/slab.h b/include/linux/slab.h
index b22a158..e17d348 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -198,6 +198,11 @@ 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
+ * @cachep: backpointer to the kmem_cache structure that hold us.
+ * @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 +211,10 @@ struct memcg_cache_params {
    struct {
        struct mem_cgroup *memcg;
        struct kmem_cache *root_cache;
+    struct kmem_cache *cachep;
+    bool dead;
+    atomic_t nr_pages;
+    struct work_struct destroy;
    };
};
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 4d2a01f..f744305 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -2949,6 +2949,31 @@ 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);
+    cachep = p->cachep;
+
+    if (!atomic_read(&cachep->memcg_params->nr_pages))
+        kmem_cache_destroy(cachep);
+}
+static DECLARE_WORK(kmem_cache_destroy_work, kmem_cache_destroy_work_func);
+
+void mem_cgroup_destroy_cache(struct kmem_cache *cachep)
+{
+    if (!cachep->memcg_params->dead)
+        return;
+
+

```



```

+{
+}
#endif /* CONFIG_MEMCG_KMEM */

#ifdef CONFIG_TRANSPARENT_HUGEPAGE
@@ -5872,6 +5918,7 @@ 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 03952c4..39127f6 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->pfnmemalloc)
        SetPageSlabPfnmemalloc(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_accounted_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 @@
#ifdef 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 257e130..e98fdf0 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->pfnmemalloc)
@@ -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)

    __ClearPageSlabPfnmemalloc(page);
    __ClearPageSlab(page);
+
+ memcg_release_pages(s, order);
    reset_page_mapcount(page);
    if (current->reclaim_state)
        current->reclaim_state->reclaimed_slab += pages;
--
1.7.11.4

```

---