Subject: [PATCH 10/10] memcg/sl[au]b: shrink dead caches
Posted by Glauber Costa on Wed, 25 Jul 2012 14:38:21 GMT
View Forum Message <> Reply to Message

In the slub allocator, when the last object of a page goes away, we
don't necessarily free it - there is not necessarily a test for empty
page in any slab_free path.

This means that when we destroy a memcg cache that happened to be empty,
those caches may take a lot of time to go away: removing the memcg
reference won't destroy them - because there are pending references, and
the empty pages will stay there, until a shrinker is called upon for any
reason.

This patch marks all memcg caches as dead. kmem_cache_shrink is called
for the ones who are not yet dead - this will force internal cache
reorganization, and then all references to empty pages will be removed.

An unlikely branch is used to make sure this case does not affect
performance in the usual slab_free path.

The slab allocator has a time based reaper that would eventually get rid
of the objects, but we can also call it explicitly, since dead caches
are not a likely event.

[v2: also call verify_dead 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>
---
 include/linux/slab.h | 3 +++
 mm/memcontrol.c | 44 ++-
 mm/slab.c | 2 ++
 mm/slab.h | 10 ++++++++++
 mm/slub.c | 1 +
 5 files changed, 59 insertions(+), 1 deletion(-)

diff --git a/include/linux/slab.h b/include/linux/slab.h
index 3cfd784..6ca9ccb 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -182,6 +182,8 @@ unsigned int kmem_cache_size(struct kmem_cache *);
 #endif

```
 #ifdef CONFIG_MEMCG_KMEM
+#include <linux/workqueue.h>
+
 struct mem_cgroup_cache_params {
  struct mem_cgroup *memcg;
  struct kmem_cache *parent;
@@ -190,6 +192,7 @@ struct mem_cgroup_cache_params {
  atomic_t nr_pages;
  struct list_head destroyed_list; /* Used when deleting memcg cache */
  struct list_head sibling_list;
+ struct work_struct cache_shrinker;
 };
 #endif

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index d92d963..747efec 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -568,7 +568,7 @@ static char *memcg_cache_name(struct mem_cgroup *memcg, struct
kmem_cache *cache

  BUG_ON(dentry == NULL);

- name = kasprintf(GFP_KERNEL, "%s(%d:%s)",
+ name = kasprintf(GFP_KERNEL, "%s(%d:%s)dead",
      cachep->name, css_id(&memcg->css), dentry->d_name.name);

  return name;
@@ -749,12 +749,25 @@ static void disarm_kmem_keys(struct mem_cgroup *memcg)
  WARN_ON(res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0);
 }

+static void cache_shrinker_work_func(struct work_struct *work)
+{
+ struct mem_cgroup_cache_params *params;
+ struct kmem_cache *cachep;
+
+ params = container_of(work, struct mem_cgroup_cache_params,
+         cache_shrinker);
+ cachep = container_of(params, struct kmem_cache, memcg_params);
+
+ kmem_cache_shrink(cachep);
+}
+
 static DEFINE_MUTEX(memcg_cache_mutex);
 static struct kmem_cache *memcg_create_kmem_cache(struct mem_cgroup *memcg,
        struct kmem_cache *cachep)
```

```
 {
  struct kmem_cache *new_cachep;
  int idx;
+ char *name;

  BUG_ON(!memcg_kmem_enabled(memcg));

@@ -774,10 +787,21 @@ static struct kmem_cache *memcg_create_kmem_cache(struct
mem_cgroup *memcg,
   goto out;
  }

+ /*
+  * Because the cache is expected to duplicate the string,
+  * we must make sure it has opportunity to copy its full
+  * name. Only now we can remove the dead part from it
+  */
+ name = (char *)new_cachep->name;
+ if (name)
+  name[strlen(name) - 4] = '\0';
+
  mem_cgroup_get(memcg);
  memcg->slabs[idx] = new_cachep;
  new_cachep->memcg_params.memcg = memcg;
  atomic_set(&new_cachep->memcg_params.nr_pages , 0);
+ INIT_WORK(&new_cachep->memcg_params.cache_shrinker,
+    cache_shrinker_work_func);
 out:
  mutex_unlock(&memcg_cache_mutex);
  return new_cachep;
@@ -800,6 +824,21 @@ static void kmem_cache_destroy_work_func(struct work_struct *w)
  struct mem_cgroup_cache_params *p, *tmp;
  unsigned long flags;
  LIST_HEAD(del_unlocked);
+ LIST_HEAD(shrinkers);
+
+ 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);
+  if (atomic_read(&cachep->memcg_params.nr_pages) != 0)
+   list_move(&cachep->memcg_params.destroyed_list, &shrinkers);
+ }
+ spin_unlock_irqrestore(&cache_queue_lock, flags);
+
+ list_for_each_entry_safe(p, tmp, &shrinkers, destroyed_list) {
+  cachep = container_of(p, struct kmem_cache, memcg_params);
+  list_del(&cachep->memcg_params.destroyed_list);
+  kmem_cache_shrink(cachep);
```

```
+ }

  spin_lock_irqsave(&cache_queue_lock, flags);
  list_for_each_entry_safe(p, tmp, &destroyed_caches, destroyed_list) {
@@ -877,11 +916,14 @@ static void mem_cgroup_destroy_all_caches(struct mem_cgroup
*memcg)

  spin_lock_irqsave(&cache_queue_lock, flags);
  for (i = 0; i < MAX_KMEM_CACHE_TYPES; i++) {
+  char *name;
   cachep = memcg->slabs[i];
   if (!cachep)
    continue;

   cachep->memcg_params.dead = true;
+  name = (char *)cachep->name;
+  name[strlen(name)] = 'd';
   __mem_cgroup_destroy_cache(cachep);
  }
  spin_unlock_irqrestore(&cache_queue_lock, flags);
diff --git a/mm/slab.c b/mm/slab.c
index 6d8d449..68b293b 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -3558,6 +3558,8 @@ static inline void __cache_free(struct kmem_cache *cachep, void
*objp,
  }

  ac->entry[ac->avail++] = objp;
+
+ kmem_cache_verify_dead(cachep);
 }

 /**
diff --git a/mm/slab.h b/mm/slab.h
index d9df178..0e748e8 100644
--- a/mm/slab.h
+++ b/mm/slab.h
@@ -80,6 +80,12 @@ static inline void mem_cgroup_release_pages(struct kmem_cache *s, int
order)
  if (atomic_sub_and_test((1 << order), &s->memcg_params.nr_pages))
   mem_cgroup_destroy_cache(s);
 }
+
+static inline void kmem_cache_verify_dead(struct kmem_cache *s)
+{
+ if (unlikely(s->memcg_params.dead))
+  schedule_work(&s->memcg_params.cache_shrinker);
```

```
+}
 #else
 static inline bool slab_is_parent(struct kmem_cache *s, struct kmem_cache *p)
 {
@@ -98,4 +104,8 @@ static inline void mem_cgroup_bind_pages(struct kmem_cache *s, int
order)
 static inline void mem_cgroup_release_pages(struct kmem_cache *s, int order)
 {
 }
+
+static inline void kmem_cache_verify_dead(struct kmem_cache *s)
+{
+}
 #endif
diff --git a/mm/slub.c b/mm/slub.c
index fdb1a0d..55946c3 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -2582,6 +2582,7 @@ redo:
  } else
   __slab_free(s, page, x, addr);

+ kmem_cache_verify_dead(s);
 }

 void kmem_cache_free(struct kmem_cache *s, void *x)
--
1.7.10.4
```