Subject: [PATCH v3 22/28] memcg/slub: shrink dead caches
Posted by Glauber Costa on Fri, 25 May 2012 13:03:42 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.

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 | 4 +++
 include/linux/slub_def.h | 8 +++++++
 mm/memcontrol.c | 49 ++--
 mm/slub.c | 1 +
 4 files changed, 59 insertions(+), 3 deletions(-)

diff --git a/include/linux/slab.h b/include/linux/slab.h
index c81a5d3..25f073e 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -154,10 +154,14 @@ unsigned int kmem_cache_size(struct kmem_cache *);
 #endif

 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#include <linux/workqueue.h>
+
 struct mem_cgroup_cache_params {
  struct mem_cgroup *memcg;
  int id;

```
  atomic_t refcnt;
+ bool dead;
+ struct work_struct cache_shrinker;

 #ifdef CONFIG_SLAB
  /* Original cache parameters, used when creating a memcg cache */
diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index ba9c68b..c1428ee 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -135,6 +135,14 @@ static inline bool slab_is_parent(struct kmem_cache *s,
 #endif
 }

+static inline void kmem_cache_verify_dead(struct kmem_cache *cachep)
+{
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (unlikely(cachep->memcg_params.dead))
+  schedule_work(&cachep->memcg_params.cache_shrinker);
+#endif
+}
+
 /*
  * Kmalloc subsystem.
  */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index e2ba527..e2576c5 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -520,7 +520,7 @@ char *mem_cgroup_cache_name(struct mem_cgroup *memcg, struct
kmem_cache *cachep)

  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;
@@ -557,11 +557,24 @@ void mem_cgroup_release_cache(struct kmem_cache *cachep)
   ida_simple_remove(&cache_types, cachep->memcg_params.id);
 }

+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 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(!mem_cgroup_kmem_enabled(memcg));

@@ -581,10 +594,21 @@ 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.refcnt, 1);
+ INIT_WORK(&new_cachep->memcg_params.cache_shrinker,
+    cache_shrinker_work_func);
 out:
  mutex_unlock(&memcg_cache_mutex);
  return new_cachep;
@@ -607,6 +631,21 @@ 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.refcnt) != 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) {
@@ -682,12 +721,16 @@ 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;

-  if (atomic_dec_and_test(&cachep->memcg_params.refcnt))
-   __mem_cgroup_destroy_cache(cachep);
+  atomic_dec(&cachep->memcg_params.refcnt);
+  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/slub.c b/mm/slub.c
index eb0ff97..f5fc10c 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -2658,6 +2658,7 @@ redo:
  } else
   __slab_free(s, page, x, addr);

+ kmem_cache_verify_dead(s);
 }

 void kmem_cache_free(struct kmem_cache *s, void *x)
--
1.7.7.6
```