
Subject: [PATCH v3 19/28] slab: per-memcg accounting of slab caches
Posted by [Glauber Costa](#) on Fri, 25 May 2012 13:03:39 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch charges allocation of a slab object to a particular memcg.

The cache is selected with `mem_cgroup_get_kmem_cache()`, which is the biggest overhead we pay here, because it happens at all allocations. However, other than forcing a function call, this function is not very expensive, and try to return as soon as we realize we are not a memcg cache.

The charge/uncharge functions are heavier, but are only called for new page allocations.

Code is heavily inspired by Suleiman's, with adaptations to the patchset and minor simplifications by me.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Signed-off-by: Suleiman Souhlal <suleiman@google.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>

include/linux/slab_def.h | 62 +++++
mm/slab.c | 98 +++++-----
2 files changed, 151 insertions(+), 9 deletions(-)

diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
index 7c0cdd6..ea9054a 100644
--- a/include/linux/slab_def.h
+++ b/include/linux/slab_def.h
@@ -225,4 +225,66 @@ found:

```
#endif /* CONFIG_NUMA */  
  
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM  
+  
+void kmem_cache_drop_ref(struct kmem_cache *cachep);  
+  
+static inline void  
+kmem_cache_get_ref(struct kmem_cache *cachep)  
+{  
+ if (cachep->memcg_params.id == -1 &&  
+ unlikely(!atomic_add_unless(&cachep->memcg_params.refcnt, 1, 0)))
```



```

spin_lock(&l3->list_lock);

@@ -3105,8 +3126,10 @@ static int cache_grow(struct kmem_cache *cachep,
oops1:
    kmem_freepages(cachep, objp);
failed:
- if (local_flags & __GFP_WAIT)
+ if (local_flags & __GFP_WAIT) {
    local_irq_disable();
+ mem_cgroup_kmem_cache_finish_sleep(cachep);
+ }
    return 0;
}

@@ -3867,11 +3890,15 @@ static inline void __cache_free(struct kmem_cache *cachep, void
*objp,
*/
void *kmem_cache_alloc(struct kmem_cache *cachep, gfp_t flags)
{
- void *ret = __cache_alloc(cachep, flags, __builtin_return_address(0));
+ void *ret;
+
+ rcu_read_lock();
+ cachep = mem_cgroup_get_kmem_cache(cachep, flags);
+ rcu_read_unlock();
+ ret = __cache_alloc(cachep, flags, __builtin_return_address(0));

    trace_kmem_cache_alloc(_RET_IP_, ret,
        obj_size(cachep), cachep->buffer_size, flags);
-
    return ret;
}
EXPORT_SYMBOL(kmem_cache_alloc);
@@ -3882,6 +3909,10 @@ kmem_cache_alloc_trace(size_t size, struct kmem_cache *cachep,
gfp_t flags)
{
    void *ret;

+ rcu_read_lock();
+ cachep = mem_cgroup_get_kmem_cache(cachep, flags);
+ rcu_read_unlock();
+
    ret = __cache_alloc(cachep, flags, __builtin_return_address(0));

    trace_kmalloc(_RET_IP_, ret,
@@ -3894,13 +3925,17 @@ EXPORT_SYMBOL(kmem_cache_alloc_trace);
#ifdef CONFIG_NUMA
void *kmem_cache_alloc_node(struct kmem_cache *cachep, gfp_t flags, int nodeid)

```

```

{
- void *ret = __cache_alloc_node(cachep, flags, nodeid,
+ void *ret;
+
+ rcu_read_lock();
+ cachep = mem_cgroup_get_kmem_cache(cachep, flags);
+ rcu_read_unlock();
+ ret = __cache_alloc_node(cachep, flags, nodeid,
    __builtin_return_address(0));

    trace_kmem_cache_alloc_node(_RET_IP_, ret,
        obj_size(cachep), cachep->buffer_size,
        flags, nodeid);
-
    return ret;
}
EXPORT_SYMBOL(kmem_cache_alloc_node);
@@ -3913,6 +3948,9 @@ void *kmem_cache_alloc_node_trace(size_t size,
{
    void *ret;

+ rcu_read_lock();
+ cachep = mem_cgroup_get_kmem_cache(cachep, flags);
+ rcu_read_unlock();
    ret = __cache_alloc_node(cachep, flags, nodeid,
        __builtin_return_address(0));
    trace_kmalloc_node(_RET_IP_, ret,
@@ -4021,9 +4059,33 @@ void kmem_cache_free(struct kmem_cache *cachep, void *objp)

    local_irq_save(flags);
    debug_check_no_locks_freed(objp, obj_size(cachep));
+
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ {
+     struct kmem_cache *actual_cachep;
+
+     actual_cachep = virt_to_cache(objp);
+     if (actual_cachep != cachep) {
+         VM_BUG_ON(actual_cachep->memcg_params.id != -1);
+         cachep = actual_cachep;
+     }
+     /*
+      * Grab a reference so that the cache is guaranteed to stay
+      * around.
+      * If we are freeing the last object of a dead memcg cache,
+      * the kmem_cache_drop_ref() at the end of this function
+      * will end up freeing the cache.
+      */

```

```

+ kmem_cache_get_ref(cachep);
+ }
+ #endif
+
+ if (!(cachep->flags & SLAB_DEBUG_OBJECTS))
+   debug_check_no_obj_freed(objp, obj_size(cachep));
+   __cache_free(cachep, objp, __builtin_return_address(0));
+
+ kmem_cache_drop_ref(cachep);
+
+ local_irq_restore(flags);

+ trace_kmem_cache_free(_RET_IP_, objp);
@@ -4051,9 +4113,19 @@ void kfree(const void *objp)
+   local_irq_save(flags);
+   kfree_debugcheck(objp);
+   c = virt_to_cache(objp);
+
+   /*
+   * Grab a reference so that the cache is guaranteed to stay around.
+   * If we are freeing the last object of a dead memcg cache, the
+   * kmem_cache_drop_ref() at the end of this function will end up
+   * freeing the cache.
+   */
+   kmem_cache_get_ref(c);
+
+   debug_check_no_locks_freed(objp, obj_size(c));
+   debug_check_no_obj_freed(objp, obj_size(c));
+   __cache_free(c, (void *)objp, __builtin_return_address(0));
+ kmem_cache_drop_ref(c);
+   local_irq_restore(flags);
+ }
+ EXPORT_SYMBOL(kfree);
@@ -4322,6 +4394,13 @@ static void cache_reap(struct work_struct *w)
+   list_for_each_entry(searchp, &cache_chain, next) {
+     check_irq_on();

+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+   /* For memcg caches, make sure we only reap the active ones. */
+   if (searchp->memcg_params.id == -1 &&
+       !atomic_add_unless(&searchp->memcg_params.refcnt, 1, 0))
+     continue;
+ #endif
+
+   /*
+   * We only take the l3 lock if absolutely necessary and we
+   * have established with reasonable certainty that
+   @@ -4354,6 +4433,7 @@ static void cache_reap(struct work_struct *w)

```

```
    STATS_ADD_REAPED(searchhp, freed);  
    }  
next:  
+ kmem_cache_drop_ref(searchhp);  
  cond_resched();  
}  
check_irq_on();  
--
```

1.7.7.6
