
Subject: [PATCH v3 10/16] sl[au]b: Allocate objects from memcg cache
Posted by [Glauber Costa](#) on Tue, 18 Sep 2012 14:12:04 GMT
[View Forum Message](#) <> [Reply to Message](#)

We are able to match a cache allocation to a particular memcg. If the task doesn't change groups during the allocation itself - a rare event, this will give us a good picture about who is the first group to touch a cache page.

This patch uses the now available infrastructure by calling memcg_kmem_get_cache() before all the cache allocations.

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/slub_def.h | 15 ++++++-----
mm/memcontrol.c | 9 +++++++
mm/slabb.c | 6 +++++-
mm/slub.c | 5 +++--
4 files changed, 27 insertions(+), 8 deletions(-)

diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index 8bb8ad2..76bf6da 100644

--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -13,6 +13,8 @@
#include <linux/kobject.h>

#include <linux/kmemleak.h>
+#include <linux/memcontrol.h>
+#include <linux/mm.h>

enum stat_item {
 ALLOC_FASTPATH, /* Allocation from cpu slab */
@@ -209,14 +211,14 @@ static __always_inline int kmalloc_index(size_t size)
 * This ought to end up with a global pointer to the right cache
 * in kmalloc_caches.
 */
-static __always_inline struct kmem_cache *kmalloc_slab(size_t size)
+static __always_inline struct kmem_cache *kmalloc_slab(gfp_t flags, size_t size)
{
 int index = kmalloc_index(size);

```

if (index == 0)
    return NULL;

- return kmem_cache[index];
+ return memcg_kmem_get_cache(kmem_cache[index], flags);
}

void *kmem_cache_alloc(struct kmem_cache *, gfp_t);
@@ -225,7 +227,10 @@ void *__kmemalloc(size_t size, gfp_t flags);
static __always_inline void *
kmemalloc_order(size_t size, gfp_t flags, unsigned int order)
{
- void *ret = (void *) __get_free_pages(flags | __GFP_COMP, order);
+ void *ret;
+
+ flags |= (__GFP_COMP | __GFP_KMEMCG);
+ ret = (void *) __get_free_pages(flags, order);
    kmemleak_alloc(ret, size, 1, flags);
    return ret;
}
@@ -274,7 +279,7 @@ static __always_inline void *kmemalloc(size_t size, gfp_t flags)
    return kmemalloc_large(size, flags);

    if (!(flags & SLUB_DMA)) {
- struct kmem_cache *s = kmemalloc_slab(size);
+ struct kmem_cache *s = kmemalloc_slab(flags, size);

    if (!s)
        return ZERO_SIZE_PTR;
@@ -307,7 +312,7 @@ static __always_inline void *kmemalloc_node(size_t size, gfp_t flags, int
node)
{
    if (__builtin_constant_p(size) &&
        size <= SLUB_MAX_SIZE && !(flags & SLUB_DMA)) {
- struct kmem_cache *s = kmemalloc_slab(size);
+ struct kmem_cache *s = kmemalloc_slab(flags, size);

    if (!s)
        return ZERO_SIZE_PTR;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index ee982aa..0068b7d 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -477,7 +477,14 @@ struct mem_cgroup *mem_cgroup_from_css(struct
cgroup_subsys_state *s)
#include <net/sock.h>
#include <net/ip.h>

```

```

+/*
+ * A lot of the calls to the cache allocation functions are expected to be
+ * inlined by the compiler. Since the calls to memcg_kmem_get_cache are
+ * conditional to this static branch, we'll have to allow modules that does
+ * kmem_cache_alloc and the such to see this symbol as well
+ */
+ struct static_key memcg_kmem_enabled_key;
+EXPORT_SYMBOL(memcg_kmem_enabled_key);

static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, u64 size);
@@ -589,6 +596,8 @@ static struct kmem_cache *kmem_cache_dup(struct mem_cgroup
*memcg,

    new = kmem_cache_create_memcg(memcg, name, s->object_size, s->align,
        (s->flags & ~SLAB_PANIC), s->ctor, s);
+ if (new)
+ new->allocflags |= __GFP_KMEMCG;

    kfree(name);
    return new;
diff --git a/mm/slab.c b/mm/slab.c
index 18de3f6..f9c7e03 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1971,7 +1971,7 @@ static void kmem_freepages(struct kmem_cache *cachep, void *addr)
    }
    if (current->reclaim_state)
        current->reclaim_state->reclaimed_slab += nr_freed;
- free_pages((unsigned long)addr, cachep->gfporder);
+ free_accounted_pages((unsigned long)addr, cachep->gfporder);
    }

static void kmem_rcu_free(struct rcu_head *head)
@@ -3541,6 +3541,8 @@ __cache_alloc_node(struct kmem_cache *cachep, gfp_t flags, int
nodeid,
    if (slab_should_failslab(cachep, flags))
        return NULL;

+ cachep = memcg_kmem_get_cache(cachep, flags);
+
    cache_alloc_debugcheck_before(cachep, flags);
    local_irq_save(save_flags);

@@ -3626,6 +3628,8 @@ __cache_alloc(struct kmem_cache *cachep, gfp_t flags, void *caller)
    if (slab_should_failslab(cachep, flags))
        return NULL;

```

```

+ cachep = memcg_kmem_get_cache(cachep, flags);
+
  cache_alloc_debugcheck_before(cachep, flags);
  local_irq_save(save_flags);
  objp = __do_cache_alloc(cachep, flags);
diff --git a/mm/slub.c b/mm/slub.c
index a045dfc..ac79595 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1403,7 +1403,7 @@ static void __free_slab(struct kmem_cache *s, struct page *page)
  reset_page_mapcount(page);
  if (current->reclaim_state)
    current->reclaim_state->reclaimed_slab += pages;
- __free_pages(page, order);
+ __free_accounted_pages(page, order);
}

#define need_reserve_slab_rcu \
@@ -2313,6 +2313,7 @@ static __always_inline void *slab_alloc(struct kmem_cache *s,
  if (slab_pre_alloc_hook(s, gfpflags))
    return NULL;

+ s = memcg_kmem_get_cache(s, gfpflags);
redo:

/*
@@ -3454,7 +3455,7 @@ void kfree(const void *x)
  if (unlikely(!PageSlab(page))) {
    BUG_ON(!PageCompound(page));
    kmemleak_free(x);
- __free_pages(page, compound_order(page));
+ __free_accounted_pages(page, compound_order(page));
    return;
  }
  slab_free(page->slab, page, object, _RET_IP_);
--
1.7.11.4

```
