Subject: [PATCH v4 18/25] mm: Allocate kernel pages to the right memcg
Posted by Glauber Costa on Mon, 18 Jun 2012 10:28:11 GMT
View Forum Message <> Reply to Message

This patch builds on the suggestion previously given by Cristoph, with one
major difference: it still keeps the cache dispatcher and the cache duplicates.
But its internals are completely different.

I no longer mess with the cache cores when pages are allocated. (except for
destruction, that happens a bit later, but that's quite simple). All of that
is done by the page allocator, by recognizing the __GFP_SLABMEMCG flag.

The catch here is that 99% of the time, the task doing the dispatch will be
the same allocating the page. It doesn't hold only when tasks are moving around.
But that's an acceptable price to pay, at least for me. Moving around won't break,
it will at the most put us on a state where a cache has a page that is acconted
to a different cgroup. Or, if that cgroups is destroyed, not accounted to anyone.
If that ever hurts anyone, this is solvable by a reaper, or by a full cache scan
when the task moves.

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/page-flags.h | 2 +-
 include/linux/slub_def.h | 18 ++++++++++++-----
 mm/memcontrol.c | 2 ++
 mm/page_alloc.c | 13 ++++++++++-
 mm/slab.c | 4 ++++
 mm/slub.c | 1 +
 6 files changed, 33 insertions(+), 7 deletions(-)

diff --git a/include/linux/page-flags.h b/include/linux/page-flags.h
index c88d2a9..9b065d7 100644
--- a/include/linux/page-flags.h
+++ b/include/linux/page-flags.h
@@ -201,7 +201,7 @@ PAGEFLAG(Dirty, dirty) TESTSCFLAG(Dirty, dirty)
__CLEARPAGEFLAG(Dirty, dirty)
 PAGEFLAG(LRU, lru) __CLEARPAGEFLAG(LRU, lru)
 PAGEFLAG(Active, active) __CLEARPAGEFLAG(Active, active)
  TESTCLEARFLAG(Active, active)
-__PAGEFLAG(Slab, slab)
+__PAGEFLAG(Slab, slab) __TESTCLEARFLAG(Slab, slab)
 PAGEFLAG(Checked, checked) /* Used by some filesystems */

```
 PAGEFLAG(Pinned, pinned) TESTSCFLAG(Pinned, pinned) /* Xen */
 PAGEFLAG(SavePinned, savepinned);   /* Xen */
diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index 7637f3b..7183596 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 kmalloc_caches[index];
+ return mem_cgroup_get_kmem_cache(kmalloc_caches[index], flags);
 }

 void *kmem_cache_alloc(struct kmem_cache *, gfp_t);
@@ -225,7 +227,13 @@ void *__kmalloc(size_t size, gfp_t flags);
 static __always_inline void *
 kmalloc_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;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ flags |= __GFP_SLABMEMCG;
+#endif
+ ret = (void *) __get_free_pages(flags, order);
  kmemleak_alloc(ret, size, 1, flags);
  return ret;
 }
@@ -274,7 +282,7 @@ static __always_inline void *kmalloc(size_t size, gfp_t flags)
```

```
    return kmalloc_large(size, flags);

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

    if (!s)
     return ZERO_SIZE_PTR;
@@ -307,7 +315,7 @@ static __always_inline void *kmalloc_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 = kmalloc_slab(size);
+   struct kmem_cache *s = kmalloc_slab(flags, size);

   if (!s)
    return ZERO_SIZE_PTR;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 233d3bc..b47ab87 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -524,6 +524,8 @@ 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_SLABMEMCG;

  kfree(name);
  return new;
diff --git a/mm/page_alloc.c b/mm/page_alloc.c
index a884a9c..b4322b7 100644
--- a/mm/page_alloc.c
+++ b/mm/page_alloc.c
@@ -2425,6 +2425,7 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
  struct page *page = NULL;
  int migratetype = allocflags_to_migratetype(gfp_mask);
  unsigned int cpuset_mems_cookie;
+ void *handle = NULL;

  gfp_mask &= gfp_allowed_mask;

@@ -2436,6 +2437,13 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
   return NULL;

  /*
```

```
+  * Will only have any effect when __GFP_SLABMEMCG is set.
+  * This is verified in the (always inline) callee
+  */
+ if (!mem_cgroup_new_kmem_page(gfp_mask, &handle, order))
+  return NULL;
+
+ /*
   * Check the zones suitable for the gfp_mask contain at least one
   * valid zone. It's possible to have an empty zonelist as a result
   * of GFP_THISNODE and a memoryless node
@@ -2474,6 +2482,8 @@ out:
  if (unlikely(!put_mems_allowed(cpuset_mems_cookie) && !page))
   goto retry_cpuset;

+ mem_cgroup_commit_kmem_page(page, handle, order);
+
  return page;
 }
 EXPORT_SYMBOL(__alloc_pages_nodemask);
@@ -2507,7 +2517,8 @@ EXPORT_SYMBOL(get_zeroed_page);
 void __free_pages(struct page *page, unsigned int order)
 {
  if (put_page_testzero(page)) {
-  __ClearPageSlab(page);
+  if (__TestClearPageSlab(page))
+   mem_cgroup_free_kmem_page(page, order);
   if (order == 0)
    free_hot_cold_page(page, 0);
   else
diff --git a/mm/slab.c b/mm/slab.c
index e537406..3da5210 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -3313,6 +3313,8 @@ __cache_alloc_node(struct kmem_cache *cachep, gfp_t flags, int
nodeid,
  if (slab_should_failslab(cachep, flags))
   return NULL;

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

@@ -3398,6 +3400,8 @@ __cache_alloc(struct kmem_cache *cachep, gfp_t flags, void *caller)
  if (slab_should_failslab(cachep, flags))
   return NULL;

+ cachep = mem_cgroup_get_kmem_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 69c5677..77944e2 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -2304,6 +2304,7 @@ static __always_inline void *slab_alloc(struct kmem_cache *s,
  if (slab_pre_alloc_hook(s, gfpflags))
   return NULL;

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

  /*
--
1.7.10.2
```