Subject: [PATCH 3/7] per-cgroup slab caches
Posted by Glauber Costa on Tue, 21 Feb 2012 11:34:35 GMT
View Forum Message <> Reply to Message

This patch creates the infrastructure to allow us to register
per-memcg slab caches. As an example implementation, I am tracking
the dentry cache, but others will follow.

I am using an opt-in istead of opt-out system here: this means the
cache needs to explicitly register itself to be tracked by memcg.
I prefer this approach since:

1) not all caches are big enough to be relevant,
2) most of the relevant ones will involve shrinking in some way,
   and it would be better be sure they are shrinker-aware
3) some objects, like network sockets, have their very own idea
   of memory control, that goes beyond the allocation itself.

Once registered, allocations made on behalf of a task living
on a cgroup will be billed to it. It is a first-touch mechanism,
but if follows what we have today, and the cgroup infrastructure
itself. No overhead is expected in object allocation: only when
slab pages are allocated and freed, any form of billing occurs.

The allocation stays billed to a cgroup until it is destroyed.
It is kept if a task leaves the cgroup, since it is close to
impossible to efficiently map an object to a task - and the tracking
is done by pages, which contain multiple objects.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Kirill A. Shutemov <kirill@shutemov.name>
CC: Greg Thelen <gthelen@google.com>
CC: Johannes Weiner <jweiner@redhat.com>
CC: Michal Hocko <mhocko@suse.cz>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Paul Turner <pjt@google.com>
CC: Frederic Weisbecker <fweisbec@gmail.com>
CC: Pekka Enberg <penberg@kernel.org>
CC: Christoph Lameter <cl@linux.com>
---
 include/linux/memcontrol.h | 29 +++++++++++++
 include/linux/slab.h | 8 ++++
 include/linux/slub_def.h | 2 +
 mm/memcontrol.c | 93 ++
 mm/slub.c | 68 ++++++++++++++++++++++++++++++--
 5 files changed, 195 insertions(+), 5 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h

```
index 4d34356..95e7e19 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -21,12 +21,41 @@
 #define _LINUX_MEMCONTROL_H
 #include <linux/cgroup.h>
 #include <linux/vm_event_item.h>
+#include <linux/slab.h>
+#include <linux/workqueue.h>
+#include <linux/shrinker.h>

 struct mem_cgroup;
 struct page_cgroup;
 struct page;
 struct mm_struct;

+struct memcg_kmem_cache {
+ struct kmem_cache *cache;
+ struct mem_cgroup *memcg; /* Should be able to do without this */
+};
+
+struct memcg_cache_struct {
+ int index;
+ struct kmem_cache *cache;
+};
+
+enum memcg_cache_indexes {
+ CACHE_DENTRY,
+ NR_CACHES,
+};
+
+int memcg_kmem_newpage(struct mem_cgroup *memcg, struct page *page, unsigned long
pages);
+void memcg_kmem_freepage(struct mem_cgroup *memcg, struct page *page, unsigned long
pages);
+struct mem_cgroup *memcg_from_shrinker(struct shrinker *s);
+
+struct memcg_kmem_cache *memcg_cache_get(struct mem_cgroup *memcg, int index);
+void register_memcg_cache(struct memcg_cache_struct *cache);
+void memcg_slab_destroy(struct kmem_cache *cache, struct mem_cgroup *memcg);
+
+struct kmem_cache *
+kmem_cache_dup(struct mem_cgroup *memcg, struct kmem_cache *base);
+
 /* Stats that can be updated by kernel. */
 enum mem_cgroup_page_stat_item {
   MEMCG_NR_FILE_MAPPED, /* # of pages charged as file rss */
diff --git a/include/linux/slab.h b/include/linux/slab.h
```

```
index 573c809..8a372cd 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -98,6 +98,14 @@
 void __init kmem_cache_init(void);
 int slab_is_available(void);

+struct mem_cgroup;
+
+unsigned long slab_nr_pages(struct kmem_cache *s);
+
+struct kmem_cache *kmem_cache_create_cg(struct mem_cgroup *memcg,
+   const char *, size_t, size_t,
+   unsigned long,
+   void (*)(void *));
 struct kmem_cache *kmem_cache_create(const char *, size_t, size_t,
    unsigned long,
    void (*)(void *));
diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index a32bcfd..4f39fff 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -100,6 +100,8 @@ struct kmem_cache {
  struct kobject kobj; /* For sysfs */
 #endif

+ struct mem_cgroup *memcg;
+ struct kmem_cache *parent_slab; /* can be moved out of here as well */
 #ifdef CONFIG_NUMA
  /*
   * Defragmentation by allocating from a remote node.
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 26fda11..2aa35b0 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -49,6 +49,7 @@
 #include <linux/page_cgroup.h>
 #include <linux/cpu.h>
 #include <linux/oom.h>
+#include <linux/slab.h>
 #include "internal.h"
 #include <net/sock.h>
 #include <net/tcp_memcontrol.h>
@@ -302,8 +303,11 @@ struct mem_cgroup {
 #ifdef CONFIG_INET
  struct tcp_memcontrol tcp_mem;
 #endif
+ struct memcg_kmem_cache kmem_cache[NR_CACHES];
```

```
 };

+struct memcg_cache_struct *kmem_avail_caches[NR_CACHES];
+
 /* Stuffs for move charges at task migration. */
 /*
  * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
@@ -4980,6 +4984,93 @@ err_cleanup:

 }

+struct memcg_kmem_cache *memcg_cache_get(struct mem_cgroup *memcg, int index)
+{
+ return &memcg->kmem_cache[index];
+}
+
+void register_memcg_cache(struct memcg_cache_struct *cache)
+{
+ BUG_ON(kmem_avail_caches[cache->index]);
+
+ kmem_avail_caches[cache->index] = cache;
+}
+
+#define memcg_kmem(memcg) \
+ (memcg->kmem_independent_accounting ?  &memcg->kmem : &memcg->res)
+
+struct kmem_cache *
+kmem_cache_dup(struct mem_cgroup *memcg, struct kmem_cache *base)
+{
+ struct kmem_cache *s;
+ unsigned long pages;
+ struct res_counter *fail;
+ /*
+  * TODO: We should use an ida-like index here, instead
+  * of the kernel address
+  */
+ char *kname = kasprintf(GFP_KERNEL, "%s-%p", base->name, memcg);
+
+ WARN_ON(mem_cgroup_is_root(memcg));
+
+ if (!kname)
+  return NULL;
+
+ s = kmem_cache_create_cg(memcg, kname, base->size,
+     base->align, base->flags, base->ctor);
+ if (WARN_ON(!s))
+  goto out;
+
```

```
+
+ pages = slab_nr_pages(s);
+
+ if (res_counter_charge(memcg_kmem(memcg), pages << PAGE_SHIFT, &fail)) {
+  kmem_cache_destroy(s);
+  s = NULL;
+ }
+
+ mem_cgroup_get(memcg);
+out:
+ kfree(kname);
+ return s;
+}
+
+int memcg_kmem_newpage(struct mem_cgroup *memcg, struct page *page, unsigned long
pages)
+{
+ unsigned long size = pages << PAGE_SHIFT;
+ struct res_counter *fail;
+
+ return res_counter_charge(memcg_kmem(memcg), size, &fail);
+}
+
+void memcg_kmem_freepage(struct mem_cgroup *memcg, struct page *page, unsigned long
pages)
+{
+ unsigned long size = pages << PAGE_SHIFT;
+
+ res_counter_uncharge(memcg_kmem(memcg), size);
+}
+
+void memcg_create_kmem_caches(struct mem_cgroup *memcg)
+{
+ int i;
+
+ for (i = 0; i < NR_CACHES; i++) {
+  struct kmem_cache *cache;
+
+  if (!kmem_avail_caches[i] || !kmem_avail_caches[i]->cache)
+   continue;
+
+  cache = kmem_avail_caches[i]->cache;
+
+  if (mem_cgroup_is_root(memcg))
+   memcg->kmem_cache[i].cache = cache;
+  else
+   memcg->kmem_cache[i].cache = kmem_cache_dup(memcg, cache);
+  memcg->kmem_cache[i].memcg = memcg;
```

```
+ }
+}
+
+
 static struct cgroup_subsys_state * __ref
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
@@ -5039,6 +5130,8 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  if (parent)
   memcg->swappiness = mem_cgroup_swappiness(parent);
  atomic_set(&memcg->refcnt, 1);
+
+ memcg_create_kmem_caches(memcg);
  memcg->move_charge_at_immigrate = 0;
  mutex_init(&memcg->thresholds_lock);
  return &memcg->css;
diff --git a/mm/slub.c b/mm/slub.c
index 4907563..f3815ec 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -31,6 +31,8 @@
 #include <linux/stacktrace.h>

 #include <trace/events/kmem.h>
+#include <linux/cgroup.h>
+#include <linux/memcontrol.h>

 /*
  * Lock order:
@@ -1281,6 +1283,7 @@ static struct page *allocate_slab(struct kmem_cache *s, gfp_t flags, int
node)
  struct page *page;
  struct kmem_cache_order_objects oo = s->oo;
  gfp_t alloc_gfp;
+ int pages;

  flags &= gfp_allowed_mask;

@@ -1314,9 +1317,17 @@ static struct page *allocate_slab(struct kmem_cache *s, gfp_t flags,
int node)
  if (!page)
   return NULL;

+ pages = 1 << oo_order(oo);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (s->memcg && memcg_kmem_newpage(s->memcg, page, pages) < 0) {
+  __free_pages(page, oo_order(oo));
```

```
+  return NULL;
+ }
+#endif
+
  if (kmemcheck_enabled
   && !(s->flags & (SLAB_NOTRACK | DEBUG_DEFAULT_FLAGS))) {
-  int pages = 1 << oo_order(oo);

   kmemcheck_alloc_shadow(page, oo_order(oo), flags, node);

@@ -1412,6 +1423,12 @@ static void __free_slab(struct kmem_cache *s, struct page *page)
  if (current->reclaim_state)
   current->reclaim_state->reclaimed_slab += pages;
  __free_pages(page, order);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (s->memcg)
+  memcg_kmem_freepage(s->memcg, page, 1 << order);
+#endif
+
 }

 #define need_reserve_slab_rcu      \
@@ -3851,7 +3868,7 @@ static int slab_unmergeable(struct kmem_cache *s)
  return 0;
 }

-static struct kmem_cache *find_mergeable(size_t size,
+static struct kmem_cache *find_mergeable(struct mem_cgroup *memcg, size_t size,
   size_t align, unsigned long flags, const char *name,
   void (*ctor)(void *))
 {
@@ -3887,13 +3904,34 @@ static struct kmem_cache *find_mergeable(size_t size,
   if (s->size - size >= sizeof(void *))
    continue;

+  if (memcg && s->memcg != memcg)
+   continue;
+
   return s;
  }
  return NULL;
 }

-struct kmem_cache *kmem_cache_create(const char *name, size_t size,
-  size_t align, unsigned long flags, void (*ctor)(void *))
+unsigned long slab_nr_pages(struct kmem_cache *s)
+{
```

```
+ int node;
+ unsigned long nr_slabs = 0;
+
+       for_each_online_node(node) {
+               struct kmem_cache_node *n = get_node(s, node);
+
+               if (!n)
+                       continue;
+
+               nr_slabs += atomic_long_read(&n->nr_slabs);
+       }
+
+ return nr_slabs << oo_order(s->oo);
+}
+
+struct kmem_cache *
+kmem_cache_create_cg(struct mem_cgroup *memcg, const char *name, size_t size,
+       size_t align, unsigned long flags, void (*ctor)(void *))
 {
  struct kmem_cache *s;
  char *n;
@@ -3901,8 +3939,12 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size,
  if (WARN_ON(!name))
   return NULL;

+#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ WARN_ON(memcg != NULL);
+#endif
+
  down_write(&slub_lock);
- s = find_mergeable(size, align, flags, name, ctor);
+ s = find_mergeable(memcg, size, align, flags, name, ctor);
  if (s) {
   s->refcount++;
   /*
@@ -3929,6 +3971,7 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size,
   if (kmem_cache_open(s, n,
     size, align, flags, ctor)) {
    list_add(&s->list, &slab_caches);
+   s->memcg = memcg;
    if (sysfs_slab_add(s)) {
     list_del(&s->list);
     kfree(n);
@@ -3950,6 +3993,12 @@ err:
   s = NULL;
  return s;
```

```
 }
+
+struct kmem_cache *kmem_cache_create(const char *name, size_t size,
+  size_t align, unsigned long flags, void (*ctor)(void *))
+{
+ return kmem_cache_create_cg(NULL, name, size, align, flags, ctor);
+}
 EXPORT_SYMBOL(kmem_cache_create);

 #ifdef CONFIG_SMP
@@ -5239,6 +5288,15 @@ static char *create_unique_id(struct kmem_cache *s)
  if (p != name + 1)
   *p++ = '-';
  p += sprintf(p, "%07d", s->size);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ /*
+  * TODO: We should use an ida-like index here, instead
+  * of the kernel address
+  */
+ if (s->memcg)
+  p += sprintf(p, "-%08x", (u32)((u64)(s->memcg)));
+#endif
  BUG_ON(p > name + ID_STR_LENGTH - 1);
  return name;
 }
--
1.7.7.6
```