Subject: [PATCH v5 06/18] consider a memcg parameter in kmem_create_cache
Posted by Glauber Costa on Fri, 19 Oct 2012 14:20:30 GMT
View Forum Message <> Reply to Message

Allow a memcg parameter to be passed during cache creation. When the slub allocator is being used, it will only merge caches that belong to the same memcg.

Default function is created as a wrapper, passing NULL to the memcg version. We only merge caches that belong to the same memcg.

A helper is provided, memcg_css_id: because slub needs a unique cache name for sysfs. Since this is visible, but not the canonical location for slab data, the cache name is not used, the css_id should suffice.

[v2: moved to idr/ida instead of redoing the indexes]
[v3: moved call to ida_init away from cgroup creation to fix a bug]
[v4: no longer using the index mechanism]

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>
CC: Tejun Heo <tj@kernel.org>
---
include/linux/memcontrol.h | 22 ++++++++++++++++++++
include/linux/slab.h | 12 ++++++++++-
mm/memcontrol.c | 49 ++
mm/slab.h | 23 ++++++++++++++++++++---
mm/slab_common.c | 40 ++++++++++++++++++++++++++++++++++++----
mm/slub.c | 16 ++++++++++++++--
6 files changed, 143 insertions(+), 19 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index fd4ff56..4f89a45 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -28,6 +28,7 @@ struct mem_cgroup;
struct page_cgroup;
struct page;
struct mm_struct;
+struct kmem_cache;

/* Stats that can be updated by kernel. */
enum mem_cgroup_page_stat_item {

```
@@ -414,6 +415,11 @@ void __memcg_kmem_commit_charge(struct page *page,
           struct mem_cgroup *memcg, int order);
 void __memcg_kmem_uncharge_pages(struct page *page, int order);

+int memcg_css_id(struct mem_cgroup *memcg);
+int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s);
+void memcg_release_cache(struct kmem_cache *cachep);
+void memcg_cache_list_add(struct mem_cgroup *memcg, struct kmem_cache *cachep);
+
 /**
  * memcg_kmem_newpage_charge: verify if a new kmem allocation is allowed.
  * @gfp: the gfp allocation flags.
@@ -505,6 +511,22 @@ static inline void
 memcg_kmem_commit_charge(struct page *page, struct mem_cgroup *memcg, int order)
 {
 }
+
+static inline int memcg_register_cache(struct mem_cgroup *memcg,
+           struct kmem_cache *s)
+{
+ return 0;
+}
+
+static inline void memcg_release_cache(struct kmem_cache *cachep)
+{
+}
+
+static inline void memcg_cache_list_add(struct mem_cgroup *memcg,
+     struct kmem_cache *s)
+{
+ BUG();
+}
 #endif /* CONFIG_MEMCG_KMEM */
 #endif /* _LINUX_MEMCONTROL_H */

diff --git a/include/linux/slab.h b/include/linux/slab.h
index e4ea48a..b22a158 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -116,6 +116,7 @@ struct kmem_cache {
 };
 #endif

+struct mem_cgroup;
 /*
  * struct kmem_cache related prototypes
  */
@@ -125,6 +126,9 @@ int slab_is_available(void);
```

```
 struct kmem_cache *kmem_cache_create(const char *, size_t, size_t,
    unsigned long,
    void (*)(void *));
+struct kmem_cache *
+kmem_cache_create_memcg(struct mem_cgroup *, const char *, size_t, size_t,
+   unsigned long, void (*)(void *));
 void kmem_cache_destroy(struct kmem_cache *);
 int kmem_cache_shrink(struct kmem_cache *);
 void kmem_cache_free(struct kmem_cache *, void *);
@@ -193,15 +197,21 @@ unsigned int kmem_cache_size(struct kmem_cache *);
  * Child caches will hold extra metadata needed for its operation. Fields are:
  *
  * @memcg: pointer to the memcg this cache belongs to
+ * @root_cache: pointer to the global, root cache, this cache was derived from
  */
 struct memcg_cache_params {
  bool is_root_cache;
  union {
   struct kmem_cache *memcg_caches[0];
-  struct mem_cgroup *memcg;
+  struct {
+   struct mem_cgroup *memcg;
+   struct kmem_cache *root_cache;
+  };
  };
 };

+int memcg_update_all_caches(int num_memcgs);
+
 /*
  * Common kmalloc functions provided by all allocators
  */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 6a1e096..59f6d54 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -339,6 +339,12 @@ struct mem_cgroup {
 #if defined(CONFIG_MEMCG_KMEM) && defined(CONFIG_INET)
  struct tcp_memcontrol tcp_mem;
 #endif
+#if defined(CONFIG_MEMCG_KMEM)
+ /* analogous to slab_common's slab_caches list. per-memcg */
+ struct list_head memcg_slab_caches;
+ /* Not a spinlock, we can take a lot of time walking the list */
+ struct mutex slab_caches_mutex;
+#endif
 };
```

```
 /* internal only representation about the status of kmem accounting. */
@@ -2755,6 +2761,49 @@ static void memcg_uncharge_kmem(struct mem_cgroup *memcg,
u64 size)
   mem_cgroup_put(memcg);
 }

+void memcg_cache_list_add(struct mem_cgroup *memcg, struct kmem_cache *cachep)
+{
+ mutex_lock(&memcg->slab_caches_mutex);
+ list_add(&cachep->list, &memcg->memcg_slab_caches);
+ mutex_unlock(&memcg->slab_caches_mutex);
+}
+
+/*
+ * helper for acessing a memcg's index. It will be used as an index in the
+ * child cache array in kmem_cache, and also to derive its name.
+ *
+ * According to kernel/cgroup.c, needs to be rcu protected.
+ */
+int memcg_css_id(struct mem_cgroup *memcg)
+{
+ int id;
+ rcu_read_lock();
+ id = css_id(&memcg->css);
+ rcu_read_unlock();
+ return id;
+}
+
+int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s)
+{
+ size_t size = sizeof(struct memcg_cache_params);
+
+ if (!memcg_kmem_enabled())
+  return 0;
+
+ s->memcg_params = kzalloc(size, GFP_KERNEL);
+ if (!s->memcg_params)
+  return -ENOMEM;
+
+ if (memcg)
+  s->memcg_params->memcg = memcg;
+ return 0;
+}
+
+void memcg_release_cache(struct kmem_cache *s)
+{
+ kfree(s->memcg_params);
+}
```

```
+
 /*
  * We need to verify if the allocation against current->mm->owner's memcg is
  * possible for the given order. But the page is not allocated yet, so we'll
diff --git a/mm/slab.h b/mm/slab.h
index 5ee1851..c35ecce 100644
--- a/mm/slab.h
+++ b/mm/slab.h
@@ -35,12 +35,15 @@ extern struct kmem_cache *kmem_cache;
 /* Functions provided by the slab allocators */
 extern int __kmem_cache_create(struct kmem_cache *, unsigned long flags);

+struct mem_cgroup;
 #ifdef CONFIG_SLUB
-struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
- size_t align, unsigned long flags, void (*ctor)(void *));
+struct kmem_cache *
+__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
+     size_t align, unsigned long flags, void (*ctor)(void *));
 #else
-static inline struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
- size_t align, unsigned long flags, void (*ctor)(void *))
+static inline struct kmem_cache *
+__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
+     size_t align, unsigned long flags, void (*ctor)(void *))
 { return NULL; }
 #endif

@@ -98,11 +101,23 @@ static inline bool is_root_cache(struct kmem_cache *s)
 {
  return !s->memcg_params || s->memcg_params->is_root_cache;
 }
+
+static inline bool cache_match_memcg(struct kmem_cache *cachep,
+         struct mem_cgroup *memcg)
+{
+ return (is_root_cache(cachep) && !memcg) ||
+  (cachep->memcg_params->memcg == memcg);
+}
 #else
 static inline bool is_root_cache(struct kmem_cache *s)
 {
  return true;
 }

+static inline bool cache_match_memcg(struct kmem_cache *cachep,
+         struct mem_cgroup *memcg)
+{
```

```
+ return true;
+}
 #endif
 #endif
diff --git a/mm/slab_common.c b/mm/slab_common.c
index bf4b4f1..f97f7b8 100644
--- a/mm/slab_common.c
+++ b/mm/slab_common.c
@@ -18,6 +18,7 @@
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
 #include <asm/page.h>
+#include <linux/memcontrol.h>

 #include "slab.h"

@@ -27,7 +28,8 @@ DEFINE_MUTEX(slab_mutex);
 struct kmem_cache *kmem_cache;

 #ifdef CONFIG_DEBUG_VM
-static int kmem_cache_sanity_check(const char *name, size_t size)
+static int kmem_cache_sanity_check(struct mem_cgroup *memcg, const char *name,
+       size_t size)
 {
  struct kmem_cache *s = NULL;

@@ -53,7 +55,7 @@ static int kmem_cache_sanity_check(const char *name, size_t size)
    continue;
   }

-  if (!strcmp(s->name, name)) {
+  if (cache_match_memcg(s, memcg) && !strcmp(s->name, name)) {
    pr_err("%s (%s): Cache name already exists.\n",
           __func__, name);
    dump_stack();
@@ -66,7 +68,8 @@ static int kmem_cache_sanity_check(const char *name, size_t size)
  return 0;
 }
 #else
-static inline int kmem_cache_sanity_check(const char *name, size_t size)
+static inline int kmem_cache_sanity_check(struct mem_cgroup *memcg,
+       const char *name, size_t size)
 {
  return 0;
 }
@@ -97,8 +100,9 @@ static inline int kmem_cache_sanity_check(const char *name, size_t size)
  * as davem.
  */
```

```
-struct kmem_cache *kmem_cache_create(const char *name, size_t size, size_t align,
-  unsigned long flags, void (*ctor)(void *))
+struct kmem_cache *
+kmem_cache_create_memcg(struct mem_cgroup *memcg, const char *name, size_t size,
+   size_t align, unsigned long flags, void (*ctor)(void *))
 {
  struct kmem_cache *s = NULL;
  int err = 0;
@@ -106,7 +110,7 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t size,
size_t align
  get_online_cpus();
  mutex_lock(&slab_mutex);

- if (!kmem_cache_sanity_check(name, size) == 0)
+ if (!kmem_cache_sanity_check(memcg, name, size) == 0)
   goto out_locked;

  /*
@@ -117,7 +121,7 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t size,
size_t align
   */
  flags &= CACHE_CREATE_MASK;

- s = __kmem_cache_alias(name, size, align, flags, ctor);
+ s = __kmem_cache_alias(memcg, name, size, align, flags, ctor);
  if (s)
   goto out_locked;

@@ -126,6 +130,13 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size, size_t align
   s->object_size = s->size = size;
   s->align = align;
   s->ctor = ctor;
+
+  if (memcg_register_cache(memcg, s)) {
+   kmem_cache_free(kmem_cache, s);
+   err = -ENOMEM;
+   goto out_locked;
+  }
+
   s->name = kstrdup(name, GFP_KERNEL);
   if (!s->name) {
    kmem_cache_free(kmem_cache, s);
@@ -135,10 +146,11 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size, size_t align

   err = __kmem_cache_create(s, flags);
```

```
   if (!err) {
-
    s->refcount = 1;
-   list_add(&s->list, &slab_caches);
-
+   if (!memcg)
+    list_add(&s->list, &slab_caches);
+   else
+    memcg_cache_list_add(memcg, s);
   } else {
    kfree(s->name);
    kmem_cache_free(kmem_cache, s);
@@ -166,6 +178,13 @@ out_locked:

  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_memcg(NULL, name, size, align, flags, ctor);
+}
 EXPORT_SYMBOL(kmem_cache_create);

 void kmem_cache_destroy(struct kmem_cache *s)
@@ -180,6 +199,7 @@ void kmem_cache_destroy(struct kmem_cache *s)

    list_del(&s->list);

+   memcg_release_cache(s);
    kfree(s->name);
    kmem_cache_free(kmem_cache, s);
   } else {
diff --git a/mm/slub.c b/mm/slub.c
index a34548e..05aefe2 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -31,6 +31,7 @@
 #include <linux/fault-inject.h>
 #include <linux/stacktrace.h>
 #include <linux/prefetch.h>
+#include <linux/memcontrol.h>

 #include <trace/events/kmem.h>

@@ -3880,7 +3881,7 @@ 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 *))
 {
@@ -3916,17 +3917,20 @@ static struct kmem_cache *find_mergeable(size_t size,
   if (s->size - size >= sizeof(void *))
    continue;

+  if (!cache_match_memcg(s, memcg))
+   continue;
   return s;
  }
  return NULL;
 }

-struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
-  size_t align, unsigned long flags, void (*ctor)(void *))
+struct kmem_cache *
+__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
+     size_t align, unsigned long flags, void (*ctor)(void *))
 {
  struct kmem_cache *s;

- s = find_mergeable(size, align, flags, name, ctor);
+ s = find_mergeable(memcg, size, align, flags, name, ctor);
  if (s) {
   s->refcount++;
   /*
@@ -5246,6 +5250,10 @@ static char *create_unique_id(struct kmem_cache *s)
  if (p != name + 1)
   *p++ = '-';
  p += sprintf(p, "%07d", s->size);
+#ifdef CONFIG_MEMCG_KMEM
+ if (!is_root_cache(s))
+  p += sprintf(p, "-%08d", memcg_css_id(s->memcg_params->memcg));
+#endif
  BUG_ON(p > name + ID_STR_LENGTH - 1);
  return name;
 }
--
1.7.11.7
```