Subject: [PATCH v3 05/16] consider a memcg parameter in kmem_create_cache
Posted by Glauber Costa on Tue, 18 Sep 2012 14:11:59 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.

>From the memcontrol.c side, 3 helper functions are created:

1) 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.

2) mem_cgroup_register_cache: is responsible for assigning
   a unique index to each cache, and other general purpose
   setup. The index is only assigned for the root caches. All
   others are assigned index == -1.

3) mem_cgroup_release_cache: can be called from the root cache
   destruction, and will release the index for
   other caches.

We can't assign indexes until the basic slab is up and running
this is because the ida subsystem will itself call slab functions
such as kmalloc a couple of times. Because of that, we have
a late_initcall that scan all caches and register them after the
kernel is booted up. Only caches registered after that receive
their index right away.

This index mechanism was developed by Suleiman Souhlal.
Changed to a idr/ida based approach based on suggestion
from Kamezawa.

[v2: moved to idr/ida instead of redoing the indexes]
[v3: moved call to ida_init away from cgroup creation to fix a bug]

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/memcontrol.h | 20 ++++++++++++++++++
 include/linux/slab.h       |  8 ++++++++
 mm/memcontrol.c            | 28 ++++++++++++++++++++++++++++
 mm/slab.c                  |  1 +
 mm/slab.h                  | 26 ++++++++++++++++++++++----
 mm/slab_common.c           | 47 +++++++++++++++++++++++++++++++++++++++++++----------
 mm/slub.c                  | 17 ++++++++++++++---
 7 files changed, 128 insertions(+), 19 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 4ec9fd5..a5f3055 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 {
@@ -413,7 +414,26 @@ extern bool __memcg_kmem_newpage_charge(gfp_t gfp, struct
mem_cgroup **memcg,
 extern void __memcg_kmem_commit_charge(struct page *page,
                                        struct mem_cgroup *memcg, int order);
 extern void __memcg_kmem_uncharge_page(struct page *page, int order);
+extern int memcg_css_id(struct mem_cgroup *memcg);
+extern void memcg_init_kmem_cache(void);
+extern void memcg_register_cache(struct mem_cgroup *memcg,
+                                 struct kmem_cache *s);
+extern void memcg_release_cache(struct kmem_cache *cachep);
 #else
+
+static inline void memcg_init_kmem_cache(void)
+{
+}
+
+static inline void memcg_register_cache(struct mem_cgroup *memcg,
+                                        struct kmem_cache *s)
+{
+}
+
+static inline void memcg_release_cache(struct kmem_cache *cachep)
+{
+}
+
```

```
 static inline void sock_update_memcg(struct sock *sk)
 {
 }
diff --git a/include/linux/slab.h b/include/linux/slab.h
index 3152bcd..dc6daac 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 *);
@@ -337,6 +341,10 @@ extern void *__kmalloc_track_caller(size_t, gfp_t, unsigned long);
  __kmalloc(size, flags)
 #endif /* DEBUG_SLAB */

+#ifdef CONFIG_MEMCG_KMEM
+#define MAX_KMEM_CACHE_TYPES 400
+#endif
+
 #ifdef CONFIG_NUMA
 /*
  * kmalloc_node_track_caller is a special version of kmalloc_node that
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 74654f0..04851bb 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -376,6 +376,11 @@ static bool memcg_kmem_dead(struct mem_cgroup *memcg)
 {
  return test_and_clear_bit(KMEM_ACCOUNTED_DEAD, &memcg->kmem_accounted);
 }
+
+int memcg_css_id(struct mem_cgroup *memcg)
+{
+ return css_id(&memcg->css);
```

```
+}
 #endif /* CONFIG_MEMCG_KMEM */

 /* Stuffs for move charges at task migration. */
@@ -534,6 +539,29 @@ static inline bool memcg_can_account_kmem(struct mem_cgroup
*memcg)
    (memcg->kmem_accounted & (KMEM_ACCOUNTED_MASK));
 }

+struct ida cache_types;
+
+void __init memcg_init_kmem_cache(void)
+{
+ ida_init(&cache_types);
+}
+
+void memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *cachep)
+{
+ int id = -1;
+
+ if (!memcg)
+  id = ida_simple_get(&cache_types, 0, MAX_KMEM_CACHE_TYPES,
+        GFP_KERNEL);
+ cachep->memcg_params.id = id;
+}
+
+void memcg_release_cache(struct kmem_cache *cachep)
+{
+ if (cachep->memcg_params.id != -1)
+  ida_simple_remove(&cache_types, cachep->memcg_params.id);
+}
+
 /*
  * 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.c b/mm/slab.c
index 3bac667..e2cf984 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1799,6 +1799,7 @@ void __init kmem_cache_init_late(void)
    BUG();
  mutex_unlock(&slab_mutex);

+ memcg_init_kmem_cache();
  /* Done! */
  slab_state = FULL;

diff --git a/mm/slab.h b/mm/slab.h
```

```
index 7a2698b..6f2a34d 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

@@ -49,4 +52,19 @@ int __kmem_cache_shutdown(struct kmem_cache *);

 int __kmem_cache_initcall(void);

+void __init memcg_slab_register_all(void);
+#ifdef CONFIG_MEMCG_KMEM
+static inline bool cache_match_memcg(struct kmem_cache *cachep,
+         struct mem_cgroup *memcg)
+{
+ return cachep->memcg_params.memcg == memcg;
+}
+
+#else
+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 eddbb8a..8f06849 100644
--- a/mm/slab_common.c
+++ b/mm/slab_common.c
@@ -16,6 +16,7 @@
```

```
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
 #include <asm/page.h>
+#include <linux/memcontrol.h>

 #include "slab.h"

@@ -25,7 +26,8 @@ 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;

@@ -51,7 +53,7 @@ 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();
@@ -64,7 +66,8 @@ 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;
 }
@@ -95,8 +98,9 @@ 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;
```

```
@@ -104,11 +108,10 @@ 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;

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

@@ -117,6 +120,9 @@ 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;
+#ifdef CONFIG_MEMCG_KMEM
+  s->memcg_params.memcg = memcg;
+#endif
   s->name = kstrdup(name, GFP_KERNEL);
   if (!s->name) {
    kmem_cache_free(kmem_cache, s);
@@ -126,14 +132,14 @@ 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);
-
   } else {
    kfree(s->name);
    kmem_cache_free(kmem_cache, s);
   }
+  if (slab_state >= FULL)
+   memcg_register_cache(memcg, s);
  } else
   err = -ENOMEM;

@@ -157,6 +163,13 @@ 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)
@@ -171,6 +184,7 @@ void kmem_cache_destroy(struct kmem_cache *s)
    if (s->flags & SLAB_DESTROY_BY_RCU)
     rcu_barrier();

+   memcg_release_cache(s);
    kfree(s->name);
    kmem_cache_free(kmem_cache, s);
   } else {
@@ -192,6 +206,17 @@ int slab_is_available(void)

 static int __init kmem_cache_initcall(void)
 {
- return __kmem_cache_initcall();
+ int r = __kmem_cache_initcall();
+#ifdef CONFIG_MEMCG_KMEM
+ struct kmem_cache *s;
+
+ if (r)
+  return r;
+ mutex_lock(&slab_mutex);
+ list_for_each_entry(s, &slab_caches, list)
+  memcg_register_cache(NULL, s);
+ mutex_unlock(&slab_mutex);
+#endif
+ return r;
 }
 __initcall(kmem_cache_initcall);
diff --git a/mm/slub.c b/mm/slub.c
index 7ac46c6..4778548 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>
```

```
@@ -3864,7 +3865,7 @@ 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 *))
 {
@@ -3900,17 +3901,20 @@ 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++;
   /*
@@ -5230,6 +5234,10 @@ 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 (s->memcg_params.memcg)
+  p += sprintf(p, "-%08d", memcg_css_id(s->memcg_params.memcg));
+#endif
  BUG_ON(p > name + ID_STR_LENGTH - 1);
  return name;
 }
@@ -5346,6 +5354,7 @@ int __init __kmem_cache_initcall(void)
    return -ENOSYS;
   }
```

```
+ memcg_init_kmem_cache();
  slab_state = FULL;

  list_for_each_entry(s, &slab_caches, list) {
--
1.7.11.4
```