Subject: [PATCH v2 14/29] slub: consider a memcg parameter in kmem_create_cache
Posted by Glauber Costa on Fri, 11 May 2012 17:44:16 GMT
View Forum Message <> Reply to Message

Allow a memcg parameter to be passed during cache creation.
The slub allocator 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.

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 |   14 +++++++++
 include/linux/slab.h       |    6 ++++
 mm/memcontrol.c            |   27 ++++++++++++++++
 mm/slub.c                  |   67 ++++++++++++++++++++++++++++++++++++++++---
 4 files changed, 109 insertions(+), 5 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index f94efd2..99e14b9 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -26,6 +26,7 @@ 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 {
@@ -440,7 +441,20 @@ struct sock;
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 void sock_update_memcg(struct sock *sk);
 void sock_release_memcg(struct sock *sk);
+int memcg_css_id(struct mem_cgroup *memcg);
+void mem_cgroup_register_cache(struct mem_cgroup *memcg,
+          struct kmem_cache *s);
+void mem_cgroup_release_cache(struct kmem_cache *cachep);
 #else
+static inline void mem_cgroup_register_cache(struct mem_cgroup *memcg,
+          struct kmem_cache *s)
+{
+}
+
+static inline void mem_cgroup_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 dbf36b5..1386650 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -320,6 +320,12 @@ extern void *__kmalloc_track_caller(size_t, gfp_t, unsigned long);
  __kmalloc(size, flags)
 #endif /* DEBUG_SLAB */

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
```

```
+#define MAX_KMEM_CACHE_TYPES 400
+#else
+#define MAX_KMEM_CACHE_TYPES 0
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
 #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 49b1129..9327996 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -323,6 +323,11 @@ struct mem_cgroup {
 #endif
 };

+int memcg_css_id(struct mem_cgroup *memcg)
+{
+ return css_id(&memcg->css);
+}
+
 /* Stuffs for move charges at task migration. */
 /*
  * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
@@ -461,6 +466,27 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
 }
 EXPORT_SYMBOL(tcp_proto_cgroup);
 #endif /* CONFIG_INET */
+
+struct ida cache_types;
+
+void mem_cgroup_register_cache(struct mem_cgroup *memcg,
+          struct kmem_cache *cachep)
+{
+ int id = -1;
+
+ cachep->memcg_params.memcg = memcg;
+
+ if (!memcg)
+  id = ida_simple_get(&cache_types, 0, MAX_KMEM_CACHE_TYPES,
+        GFP_KERNEL);
+ cachep->memcg_params.id = id;
+}
+
+void mem_cgroup_release_cache(struct kmem_cache *cachep)
+{
+ if (cachep->memcg_params.id != -1)
+  ida_simple_remove(&cache_types, cachep->memcg_params.id);
```

```
+}
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

 static void drain_all_stock_async(struct mem_cgroup *memcg);
@@ -5053,6 +5079,7 @@ mem_cgroup_create(struct cgroup *cont)
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
   WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
         kmem_cgroup_files));
+  ida_init(&cache_types);
 #endif

   if (mem_cgroup_soft_limit_tree_init())
diff --git a/mm/slub.c b/mm/slub.c
index e606f1b..1698371 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -32,6 +32,7 @@
 #include <linux/prefetch.h>

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

 /*
  * Lock order:
@@ -186,7 +187,8 @@ static enum {
  DOWN,  /* No slab functionality available */
  PARTIAL, /* Kmem_cache_node works */
  UP,  /* Everything works but does not show up in sysfs */
- SYSFS  /* Sysfs up */
+ SYSFS,  /* Sysfs up */
+ MEMCG  /* Memcg registered */
 } slab_state = DOWN;

 /* A list of all slab caches on the system */
@@ -3193,6 +3195,11 @@ void kmem_cache_destroy(struct kmem_cache *s)
  s->refcount--;
  if (!s->refcount) {
   list_del(&s->list);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+  /* Not a memcg cache */
+  if (s->memcg_params.id != -1)
+   mem_cgroup_release_cache(s);
+#endif
   up_write(&slub_lock);
   if (kmem_cache_close(s)) {
    printk(KERN_ERR "SLUB %s: %s called for cache that "
@@ -3860,6 +3867,33 @@ void __init kmem_cache_init_late(void)
 {
```

```
 }

+static int __init memcg_slab_register_all(void)
+{
+ struct kmem_cache *s;
+ int i;
+
+ for (i = 0; i < SLUB_PAGE_SHIFT; i++) {
+  struct kmem_cache *s;
+  s = kmalloc_caches[i];
+  if (s)
+   mem_cgroup_register_cache(NULL, s);
+  s = kmalloc_dma_caches[i];
+  if (s)
+   mem_cgroup_register_cache(NULL, s);
+ }
+
+
+
+ down_write(&slub_lock);
+ list_for_each_entry(s, &slab_caches, list)
+  mem_cgroup_register_cache(NULL, s);
+
+ slab_state = MEMCG;
+ up_write(&slub_lock);
+ return 0;
+}
+late_initcall(memcg_slab_register_all);
+
 /*
  * Find a mergeable slab cache
  */
@@ -3880,7 +3914,7 @@ 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 *))
 {
@@ -3916,13 +3950,19 @@ static struct kmem_cache *find_mergeable(size_t size,
   if (s->size - size >= sizeof(void *))
    continue;

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+  if (memcg && s->memcg_params.memcg != memcg)
+   continue;
```

```
+#endif
+
   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 *))
+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;
  char *n;
@@ -3930,8 +3970,12 @@ 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++;
   /*
@@ -3959,6 +4003,8 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size,
     size, align, flags, ctor)) {
    list_add(&s->list, &slab_caches);
    up_write(&slub_lock);
+   if (slab_state >= MEMCG)
+    mem_cgroup_register_cache(memcg, s);
    if (sysfs_slab_add(s)) {
     down_write(&slub_lock);
     list_del(&s->list);
@@ -3980,6 +4026,12 @@ err:
  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);

 #ifdef CONFIG_SMP
@@ -5273,6 +5325,11 @@ 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
+ 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;
 }
--
1.7.7.6
```