Subject: [PATCH v2 15/29] slab: pass memcg parameter to kmem_cache_create
Posted by Glauber Costa on Fri, 11 May 2012 17:44:17 GMT
View Forum Message <> Reply to Message

Allow a memcg parameter to be passed during cache creation.

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

This code was mostly written by Suleiman Souhlal and
only adapted to my patchset, plus a couple of simplifications

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>

```
---
 mm/slab.c |   74 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++------
 1 files changed, 66 insertions(+), 8 deletions(-)

diff --git a/mm/slab.c b/mm/slab.c
index 56f2ba8..d05a326 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -502,6 +502,9 @@ EXPORT_SYMBOL(slab_buffer_size);
 #define SLAB_MAX_ORDER_LO 0
 static int slab_max_order = SLAB_MAX_ORDER_LO;
 static bool slab_max_order_set __initdata;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static int slab_memcg_init __read_mostly;
+#endif

 /*
  * Functions for storing/retrieving the cachep and or slab from the page
@@ -2288,14 +2291,15 @@ static int __init_refok setup_cpu_cache(struct kmem_cache
*cachep, gfp_t gfp)
  * cacheline.  This can be beneficial if you're counting cycles as closely
  * as davem.
  */
-struct kmem_cache *
-kmem_cache_create (const char *name, size_t size, size_t align,
- unsigned long flags, void (*ctor)(void *))
+static struct kmem_cache *
+__kmem_cache_create(struct mem_cgroup *memcg, const char *name, size_t size,
```

```
+      size_t align, unsigned long flags, void (*ctor)(void *))
 {
- size_t left_over, slab_size, ralign;
+ size_t left_over, orig_align, ralign, slab_size;
  struct kmem_cache *cachep = NULL, *pc;
  gfp_t gfp;

+ orig_align = align;
  /*
   * Sanity checks... these are all serious usage bugs.
   */
@@ -2312,7 +2316,6 @@ kmem_cache_create (const char *name, size_t size, size_t align,
   */
  if (slab_is_available()) {
   get_online_cpus();
-  mutex_lock(&cache_chain_mutex);
  }

  list_for_each_entry(pc, &cache_chain, next) {
@@ -2332,9 +2335,9 @@ kmem_cache_create (const char *name, size_t size, size_t align,
    continue;
   }

-  if (!strcmp(pc->name, name)) {
+  if (!strcmp(pc->name, name) && !memcg) {
    printk(KERN_ERR
-          "kmem_cache_create: duplicate cache %s\n", name);
+   "kmem_cache_create: duplicate cache %s\n", name);
    dump_stack();
    goto oops;
   }
@@ -2435,6 +2438,9 @@ kmem_cache_create (const char *name, size_t size, size_t align,
  cachep->nodelists = (struct kmem_list3 **)&cachep->array[nr_cpu_ids];

  set_obj_size(cachep, size);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ cachep->memcg_params.orig_align = orig_align;
+#endif
 #if DEBUG

  /*
@@ -2544,6 +2550,11 @@ kmem_cache_create (const char *name, size_t size, size_t align,
  cachep->ctor = ctor;
  cachep->name = kstrdup(name, GFP_KERNEL);

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (slab_memcg_init)
+  mem_cgroup_register_cache(memcg, cachep);
```

```
+#endif
+
  if (setup_cpu_cache(cachep, gfp)) {
   __kmem_cache_destroy(cachep);
   cachep = NULL;
@@ -2567,13 +2578,54 @@ oops:
   panic("kmem_cache_create(): failed to create slab `%s'\n",
         name);
  if (slab_is_available()) {
-  mutex_unlock(&cache_chain_mutex);
   put_online_cpus();
  }
  return cachep;
 }
+
+struct kmem_cache *
+kmem_cache_create(const char *name, size_t size, size_t align,
+    unsigned long flags, void (*ctor)(void *))
+{
+ struct kmem_cache *cachep;
+
+ mutex_lock(&cache_chain_mutex);
+ cachep = __kmem_cache_create(NULL, name, size, align, flags, ctor);
+ mutex_unlock(&cache_chain_mutex);
+
+ return cachep;
+}
 EXPORT_SYMBOL(kmem_cache_create);

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static int __init memcg_slab_register_all(void)
+{
+ struct kmem_cache *cachep;
+ struct cache_sizes *sizes;
+
+ mem_cgroup_register_cache(NULL, &cache_cache);
+
+ sizes = malloc_sizes;
+
+ while (sizes->cs_size != ULONG_MAX) {
+  if (sizes->cs_cachep)
+   mem_cgroup_register_cache(NULL, sizes->cs_cachep);
+  if (sizes->cs_dmacachep)
+   mem_cgroup_register_cache(NULL, sizes->cs_dmacachep);
+  sizes++;
+ }
+
+ mutex_lock(&cache_chain_mutex);
```

```
+ list_for_each_entry(cachep, &cache_chain, next)
+  mem_cgroup_register_cache(NULL, cachep);
+
+ slab_memcg_init = 1;
+ mutex_unlock(&cache_chain_mutex);
+ return 0;
+}
+late_initcall(memcg_slab_register_all);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
 #if DEBUG
 static void check_irq_off(void)
 {
@@ -2768,6 +2820,12 @@ void kmem_cache_destroy(struct kmem_cache *cachep)
  if (unlikely(cachep->flags & SLAB_DESTROY_BY_RCU))
   rcu_barrier();

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ /* Not a memcg cache */
+ if (cachep->memcg_params.id != -1)
+  mem_cgroup_release_cache(cachep);
+#endif
+
  __kmem_cache_destroy(cachep);
  mutex_unlock(&cache_chain_mutex);
  put_online_cpus();
--
1.7.7.6
```