Subject: [PATCH v3 12/28] slab: pass memcg parameter to kmem_cache_create
Posted by Glauber Costa on Fri, 25 May 2012 13:03:32 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

[v3: get_online_cpus need to be outside slab mutex.]
[also, register all caches created before FULL state]

Signed-off-by: Glauber Costa <glommer@parallels.com>
Signed-off-by: Suleiman Souhlal <suleiman@google.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>
---
 include/linux/slab_def.h | 7 ++++
 mm/slab.c | 79 ++++++++++++++++++++++++++++++++----------
 2 files changed, 69 insertions(+), 17 deletions(-)

diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
index 06e4a3e..7c0cdd6 100644
--- a/include/linux/slab_def.h
+++ b/include/linux/slab_def.h
@@ -102,6 +102,13 @@ struct kmem_cache {
    */
 };

+static inline void store_orig_align(struct kmem_cache *cachep, int orig_align)
+{
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ cachep->memcg_params.orig_align = orig_align;
+#endif
+}
+
 /* Size description struct for general caches. */
 struct cache_sizes {
   size_t   cs_size;
diff --git a/mm/slab.c b/mm/slab.c
index 41345f6..8bff32a1 100644

```
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1729,6 +1729,31 @@ void __init kmem_cache_init_late(void)
   */
 }

+static int __init memcg_slab_register_all(void)
+{
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ struct kmem_cache *cachep;
+ struct cache_sizes *sizes;
+
+ 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);
+
+ mutex_unlock(&cache_chain_mutex);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+ return 0;
+}
+
 static int __init cpucache_init(void)
 {
  int cpu;
@@ -1739,6 +1764,8 @@ static int __init cpucache_init(void)
  for_each_online_cpu(cpu)
   start_cpu_timer(cpu);

+ memcg_slab_register_all();
+
  /* Done! */
  g_cpucache_up = FULL;
  return 0;
@@ -2287,14 +2314,15 @@ 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.
   */
@@ -2305,15 +2333,6 @@ kmem_cache_create (const char *name, size_t size, size_t align,
   BUG();
  }

- /*
-  * We use cache_chain_mutex to ensure a consistent view of
-  * cpu_online_mask as well.  Please see cpuup_callback
-  */
- if (slab_is_available()) {
-  get_online_cpus();
-  mutex_lock(&cache_chain_mutex);
- }
-
  list_for_each_entry(pc, &cache_chain, next) {
   char tmp;
   int res;
@@ -2331,7 +2350,7 @@ kmem_cache_create (const char *name, size_t size, size_t align,
    continue;
   }

-  if (!strcmp(pc->name, name)) {
+  if (!memcg && !strcmp(pc->name, name)) {
    printk(KERN_ERR
           "kmem_cache_create: duplicate cache %s\n", name);
    dump_stack();
@@ -2434,6 +2453,8 @@ 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);
+
+ store_orig_align(cachep, orig_align);
 #if DEBUG
```

```
  /*
@@ -2543,7 +2564,12 @@ kmem_cache_create (const char *name, size_t size, size_t align,
   cachep->ctor = ctor;
   cachep->name = name;

+ if (g_cpucache_up >= FULL)
+   mem_cgroup_register_cache(memcg, cachep);
+
+
   if (setup_cpu_cache(cachep, gfp)) {
+   mem_cgroup_release_cache(cachep);
     __kmem_cache_destroy(cachep);
     cachep = NULL;
     goto oops;
@@ -2565,10 +2591,27 @@ oops:
   if (!cachep && (flags & SLAB_PANIC))
     panic("kmem_cache_create(): failed to create slab `%s'\n",
           name);
- if (slab_is_available()) {
-   mutex_unlock(&cache_chain_mutex);
+ 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;
+
+ /*
+  * We use cache_chain_mutex to ensure a consistent view of
+  * cpu_online_mask as well.  Please see cpuup_callback
+  */
+ if (slab_is_available())
+   get_online_cpus();
+ mutex_lock(&cache_chain_mutex);
+ cachep = __kmem_cache_create(NULL, name, size, align, flags, ctor);
+ mutex_unlock(&cache_chain_mutex);
+ if (slab_is_available())
     put_online_cpus();
- }
+
   return cachep;
  }
  EXPORT_SYMBOL(kmem_cache_create);
@@ -2767,6 +2810,8 @@ void kmem_cache_destroy(struct kmem_cache *cachep)
   if (unlikely(cachep->flags & SLAB_DESTROY_BY_RCU))
```

```
   rcu_barrier();

+ mem_cgroup_release_cache(cachep);
+
  __kmem_cache_destroy(cachep);
  mutex_unlock(&cache_chain_mutex);
  put_online_cpus();
--
1.7.7.6
```