Subject: [PATCH v3 14/28] slab: create duplicate cache
Posted by Glauber Costa on Fri, 25 May 2012 13:03:34 GMT
View Forum Message <> Reply to Message

This patch provides kmem_cache_dup(), that duplicates
a cache for a memcg, preserving its creation properties.
Object size, alignment and flags are all respected.
An exception is the SLAB_PANIC flag, since cache creation
inside a memcg should not be fatal.

This code is mostly written by Suleiman Souhlal,
with some adaptations and simplifications by me.

[ v3: add get_online cpus before the slab mutex ]

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>
---
 mm/slab.c | 44 ++++++++++++++++++++++++++++++++++++++++++++
 1 files changed, 44 insertions(+), 0 deletions(-)

diff --git a/mm/slab.c b/mm/slab.c
index 8bff32a1..e2227de 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -301,6 +301,8 @@ static void free_block(struct kmem_cache *cachep, void **objpp, int len,
    int node);
 static int enable_cpucache(struct kmem_cache *cachep, gfp_t gfp);
 static void cache_reap(struct work_struct *unused);
+static int do_tune_cpucache(struct kmem_cache *cachep, int limit,
+       int batchcount, int shared, gfp_t gfp);

 /*
  * This function must be completely optimized away if a constant is passed to
@@ -2616,6 +2618,42 @@ kmem_cache_create(const char *name, size_t size, size_t align,
 }
 EXPORT_SYMBOL(kmem_cache_create);

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+struct kmem_cache *kmem_cache_dup(struct mem_cgroup *memcg,
+      struct kmem_cache *cachep)
+{
+ struct kmem_cache *new;

```
+ unsigned long flags;
+ char *name;
+
+ name = mem_cgroup_cache_name(memcg, cachep);
+ if (!name)
+  return NULL;
+
+ flags = cachep->flags & ~(SLAB_PANIC|CFLGS_OFF_SLAB);
+
+ get_online_cpus();
+ mutex_lock(&cache_chain_mutex);
+ new = __kmem_cache_create(memcg, name, obj_size(cachep),
+     cachep->memcg_params.orig_align, flags, cachep->ctor);
+
+ if (new == NULL) {
+  kfree(name);
+  goto out;
+ }
+
+ if ((cachep->limit != new->limit) ||
+     (cachep->batchcount != new->batchcount) ||
+     (cachep->shared != new->shared))
+  do_tune_cpucache(new, cachep->limit, cachep->batchcount,
+      cachep->shared, GFP_KERNEL);
+out:
+ mutex_unlock(&cache_chain_mutex);
+ put_online_cpus();
+ return new;
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
 #if DEBUG
 static void check_irq_off(void)
 {
@@ -2811,6 +2849,12 @@ void kmem_cache_destroy(struct kmem_cache *cachep)
   rcu_barrier();

  mem_cgroup_release_cache(cachep);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ /* memcg cache: free the name string. Doing it here saves us
+  * a pointer to it outside the slab code */
+ if (cachep->memcg_params.id == -1)
+  kfree(cachep->name);
+#endif

  __kmem_cache_destroy(cachep);
  mutex_unlock(&cache_chain_mutex);
--
```

1.7.7.6