Subject: [PATCH v3 13/28] slub: create duplicate cache
Posted by Glauber Costa on Fri, 25 May 2012 13:03:33 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.

When a duplicate cache is created, the parent cache cannot
be destructed during the child lifetime. To assure this,
its reference count is increased if the cache creation
succeeds.

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 | 2 ++
 include/linux/slab.h | 2 ++
 mm/memcontrol.c | 17 ++++++++++++++++
 mm/slub.c | 32 ++++++++++++++++++++++++++++++++
 4 files changed, 53 insertions(+), 0 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 99e14b9..f93021a 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -445,6 +445,8 @@ 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);
+extern char *mem_cgroup_cache_name(struct mem_cgroup *memcg,
+   struct kmem_cache *cachep);
 #else
 static inline void mem_cgroup_register_cache(struct mem_cgroup *memcg,
                struct kmem_cache *s)
diff --git a/include/linux/slab.h b/include/linux/slab.h
index 1386650..e73ef71 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -322,6 +322,8 @@ extern void *__kmalloc_track_caller(size_t, gfp_t, unsigned long);

 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #define MAX_KMEM_CACHE_TYPES 400

```
+extern struct kmem_cache *kmem_cache_dup(struct mem_cgroup *memcg,
+      struct kmem_cache *cachep);
 #else
 #define MAX_KMEM_CACHE_TYPES 0
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index dacd1fb..4689034 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -467,6 +467,23 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
 EXPORT_SYMBOL(tcp_proto_cgroup);
 #endif /* CONFIG_INET */

+char *mem_cgroup_cache_name(struct mem_cgroup *memcg, struct kmem_cache *cachep)
+{
+ char *name;
+ struct dentry *dentry;
+
+ rcu_read_lock();
+ dentry = rcu_dereference(memcg->css.cgroup->dentry);
+ rcu_read_unlock();
+
+ BUG_ON(dentry == NULL);
+
+ name = kasprintf(GFP_KERNEL, "%s(%d:%s)",
+      cachep->name, css_id(&memcg->css), dentry->d_name.name);
+
+ return name;
+}
+
 struct ida cache_types;

 void mem_cgroup_register_cache(struct mem_cgroup *memcg,
diff --git a/mm/slub.c b/mm/slub.c
index d79740c..0eb9e72 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -4002,6 +4002,38 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size,
 }
 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 *s)
+{
+ char *name;
+ struct kmem_cache *new;
```

```
+
+ name = mem_cgroup_cache_name(memcg, s);
+ if (!name)
+  return NULL;
+
+ new = kmem_cache_create_memcg(memcg, name, s->objsize, s->align,
+          (s->allocflags & ~SLAB_PANIC), s->ctor);
+
+ /*
+  * We increase the reference counter in the parent cache, to
+  * prevent it from being deleted. If kmem_cache_destroy() is
+  * called for the root cache before we call it for a child cache,
+  * it will be queued for destruction when we finally drop the
+  * reference on the child cache.
+  */
+ if (new) {
+  down_write(&slub_lock);
+  s->refcount++;
+  up_write(&slub_lock);
+ }
+ /* slub internals is expected to have held a copy of it */
+ kfree(name);
+ return new;
+}
+#endif
+
 #ifdef CONFIG_SMP
 /*
  * Use the cpu notifier to insure that the cpu slabs are flushed when
--
1.7.7.6
```