Subject: [PATCH v5 16/18] slab: propagate tunables values
Posted by Glauber Costa on Fri, 19 Oct 2012 14:20:40 GMT
View Forum Message <> Reply to Message

SLAB allows us to tune a particular cache behavior with tunables.
When creating a new memcg cache copy, we'd like to preserve any tunables
the parent cache already had.

This could be done by an explicit call to do_tune_cpucache() after the
cache is created. But this is not very convenient now that the caches are
created from common code, since this function is SLAB-specific.

Another method of doing that is taking advantage of the fact that
do_tune_cpucache() is always called from enable_cpucache(), which is
called at cache initialization. We can just preset the values, and
then things work as expected.

It can also happen that a root cache has its tunables updated during
normal system operation. In this case, we will propagate the change to
all caches that are already active.

This change will require us to move the assignment of root_cache in
memcg_params a bit earlier. We need this to be already set - which
memcg_kmem_register_cache will do - when we reach __kmem_cache_create()

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>
CC: Tejun Heo <tj@kernel.org>
---
 include/linux/memcontrol.h | 8 +++++---
 include/linux/slab.h | 2 +-
 mm/memcontrol.c | 10 ++++++----
 mm/slab.c | 44 ++---
 mm/slab.h | 12 ++++++++++++
 mm/slab_common.c | 7 +++++---
 6 files changed, 69 insertions(+), 14 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 14def0b..9da87ff 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -421,7 +421,8 @@ void __memcg_kmem_commit_charge(struct page *page,
 void __memcg_kmem_uncharge_pages(struct page *page, int order);

```
 int memcg_css_id(struct mem_cgroup *memcg);
-int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s);
+int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s,
+     struct kmem_cache *root_cache);
 void memcg_release_cache(struct kmem_cache *cachep);
 void memcg_cache_list_add(struct mem_cgroup *memcg, struct kmem_cache *cachep);

@@ -564,8 +565,9 @@ memcg_kmem_commit_charge(struct page *page, struct mem_cgroup
*memcg, int order)
 {
 }

-static inline int memcg_register_cache(struct mem_cgroup *memcg,
-           struct kmem_cache *s)
+static inline int
+memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s,
+       struct kmem_cache *root_cache)
 {
  return 0;
 }
diff --git a/include/linux/slab.h b/include/linux/slab.h
index b521426..f8db4e1 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -128,7 +128,7 @@ struct kmem_cache *kmem_cache_create(const char *, size_t, size_t,
    void (*)(void *));
 struct kmem_cache *
 kmem_cache_create_memcg(struct mem_cgroup *, const char *, size_t, size_t,
-   unsigned long, void (*)(void *));
+   unsigned long, void (*)(void *), struct kmem_cache *);
 void kmem_cache_destroy(struct kmem_cache *);
 int kmem_cache_shrink(struct kmem_cache *);
 void kmem_cache_free(struct kmem_cache *, void *);
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index e7f3458..960d758 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -2939,7 +2939,8 @@ int memcg_update_cache_size(struct kmem_cache *s, int
num_groups)
  return 0;
 }

-int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s)
+int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s,
+     struct kmem_cache *root_cache)
 {
  size_t size = sizeof(struct memcg_cache_params);
```

```
@@ -2953,8 +2954,10 @@ int memcg_register_cache(struct mem_cgroup *memcg, struct
kmem_cache *s)
  if (!s->memcg_params)
   return -ENOMEM;

- if (memcg)
+ if (memcg) {
   s->memcg_params->memcg = memcg;
+  s->memcg_params->root_cache = root_cache;
+ }
  return 0;
 }

@@ -3098,7 +3101,7 @@ static struct kmem_cache *kmem_cache_dup(struct mem_cgroup
*memcg,
   return NULL;

  new = kmem_cache_create_memcg(memcg, name, s->object_size, s->align,
-         (s->flags & ~SLAB_PANIC), s->ctor);
+         (s->flags & ~SLAB_PANIC), s->ctor, s);

  if (new)
   new->allocflags |= __GFP_KMEMCG;
@@ -3146,7 +3149,6 @@ static struct kmem_cache *memcg_create_kmem_cache(struct
mem_cgroup *memcg,
  cachep->memcg_params->memcg_caches[idx] = new_cachep;
  wmb(); /* the readers won't lock, make sure everybody sees it */
  new_cachep->memcg_params->memcg = memcg;
- new_cachep->memcg_params->root_cache = cachep;
  atomic_set(&new_cachep->memcg_params->nr_pages , 0);
 out:
  mutex_unlock(&memcg_cache_mutex);
diff --git a/mm/slab.c b/mm/slab.c
index f29b43e4..a751033 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -4089,7 +4089,7 @@ static void do_ccupdate_local(void *info)
 }

 /* Always called with the slab_mutex held */
-static int do_tune_cpucache(struct kmem_cache *cachep, int limit,
+static int __do_tune_cpucache(struct kmem_cache *cachep, int limit,
     int batchcount, int shared, gfp_t gfp)
 {
  struct ccupdate_struct *new;
@@ -4132,12 +4132,48 @@ static int do_tune_cpucache(struct kmem_cache *cachep, int limit,
  return alloc_kmemlist(cachep, gfp);
```

```
 }

+static int do_tune_cpucache(struct kmem_cache *cachep, int limit,
+    int batchcount, int shared, gfp_t gfp)
+{
+ int ret;
+ struct kmem_cache *c = NULL;
+ int i = 0;
+
+ ret = __do_tune_cpucache(cachep, limit, batchcount, shared, gfp);
+
+ if (slab_state < FULL)
+  return ret;
+
+ if ((ret < 0) || !is_root_cache(cachep))
+  return ret;
+
+ for_each_memcg_cache_index(i) {
+  c = cache_from_memcg(cachep, i);
+  if (c)
+   /* return value determined by the parent cache only */
+   __do_tune_cpucache(c, limit, batchcount, shared, gfp);
+ }
+
+ return ret;
+}
+
 /* Called with slab_mutex held always */
 static int enable_cpucache(struct kmem_cache *cachep, gfp_t gfp)
 {
  int err;
- int limit, shared;
+ int limit = 0;
+ int shared = 0;
+ int batchcount = 0;
+
+ if (!is_root_cache(cachep)) {
+  struct kmem_cache *root = memcg_root_cache(cachep);
+  limit = root->limit;
+  shared = root->shared;
+  batchcount = root->batchcount;
+ }

+ if (limit && shared && batchcount)
+  goto skip_setup;
  /*
   * The head array serves three purposes:
   * - create a LIFO ordering, i.e. return objects that are cache-warm
```

```
@@ -4179,7 +4215,9 @@ static int enable_cpucache(struct kmem_cache *cachep, gfp_t gfp)
  if (limit > 32)
   limit = 32;
 #endif
- err = do_tune_cpucache(cachep, limit, (limit + 1) / 2, shared, gfp);
+ batchcount = (limit + 1) / 2;
+skip_setup:
+ err = do_tune_cpucache(cachep, limit, batchcount, shared, gfp);
  if (err)
   printk(KERN_ERR "enable_cpucache failed for %s, error %d.\n",
         cachep->name, -err);
diff --git a/mm/slab.h b/mm/slab.h
index c60f649..bf599fa 100644
--- a/mm/slab.h
+++ b/mm/slab.h
@@ -148,6 +148,13 @@ static inline struct kmem_cache *cache_from_memcg(struct
kmem_cache *s, int idx)
 {
  return s->memcg_params->memcg_caches[idx];
 }
+
+static inline struct kmem_cache *memcg_root_cache(struct kmem_cache *s)
+{
+ if (is_root_cache(s))
+  return s;
+ return s->memcg_params->root_cache;
+}
 #else
 static inline bool is_root_cache(struct kmem_cache *s)
 {
@@ -183,5 +190,10 @@ static inline struct kmem_cache *cache_from_memcg(struct
kmem_cache *s, int idx)
 {
  return NULL;
 }
+
+static inline struct kmem_cache *memcg_root_cache(struct kmem_cache *s)
+{
+ return s;
+}
 #endif
 #endif
diff --git a/mm/slab_common.c b/mm/slab_common.c
index 7e6d503..0dc0c41 100644
--- a/mm/slab_common.c
+++ b/mm/slab_common.c
@@ -127,7 +127,8 @@ out:
```

```
 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 *))
+   size_t align, unsigned long flags, void (*ctor)(void *),
+   struct kmem_cache *parent_cache)
 {
  struct kmem_cache *s = NULL;
  int err = 0;
@@ -156,7 +157,7 @@ kmem_cache_create_memcg(struct mem_cgroup *memcg, const char
*name, size_t size,
   s->align = align;
   s->ctor = ctor;

-  if (memcg_register_cache(memcg, s)) {
+  if (memcg_register_cache(memcg, s, parent_cache)) {
    kmem_cache_free(kmem_cache, s);
    err = -ENOMEM;
    goto out_locked;
@@ -208,7 +209,7 @@ 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);
+ return kmem_cache_create_memcg(NULL, name, size, align, flags, ctor, NULL);
 }
 EXPORT_SYMBOL(kmem_cache_create);

--
1.7.11.7
```