Subject: [PATCH v5 17/18] slub: slub-specific propagation changes.
Posted by Glauber Costa on Fri, 19 Oct 2012 14:20:41 GMT
View Forum Message <> Reply to Message

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

This can be done by tapping into the store attribute function provided
by the allocator. We of course don't need to mess with read-only
fields. Since the attributes can have multiple types and are stored
internally by sysfs, the best strategy is to issue a ->show() in the
root cache, and then ->store() in the memcg cache.

The drawback of that, is that sysfs can allocate up to a page in
buffering for show(), that we are likely not to need, but also can't
guarantee. To avoid always allocating a page for that, we can update the
caches at store time with the maximum attribute size ever stored to the
root cache. We will then get a buffer big enough to hold it. The
corolary to this, is that if no stores happened, nothing will be
propagated.

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.

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/slub_def.h | 1 +
 mm/slub.c | 76 ++-
 2 files changed, 76 insertions(+), 1 deletion(-)

diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index ed330df..f41acb9 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -105,6 +105,7 @@ struct kmem_cache {
 #endif
 #ifdef CONFIG_MEMCG_KMEM
  struct memcg_cache_params *memcg_params;
+ int max_attr_size; /* for propagation, maximum size of a stored attr */

```
 #endif

 #ifdef CONFIG_NUMA
diff --git a/mm/slub.c b/mm/slub.c
index b5b970b..41c3caf 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -203,13 +203,14 @@ enum track_item { TRACK_ALLOC, TRACK_FREE };
 static int sysfs_slab_add(struct kmem_cache *);
 static int sysfs_slab_alias(struct kmem_cache *, const char *);
 static void sysfs_slab_remove(struct kmem_cache *);
-
+static void memcg_propagate_slab_attrs(struct kmem_cache *s);
 #else
 static inline int sysfs_slab_add(struct kmem_cache *s) { return 0; }
 static inline int sysfs_slab_alias(struct kmem_cache *s, const char *p)
        { return 0; }
 static inline void sysfs_slab_remove(struct kmem_cache *s) { }

+static inline void memcg_propagate_slab_attrs(struct kmem_cache *s) { }
 #endif

 static inline void stat(const struct kmem_cache *s, enum stat_item si)
@@ -3973,6 +3974,7 @@ int __kmem_cache_create(struct kmem_cache *s, unsigned long
flags)
  if (err)
   return err;

+ memcg_propagate_slab_attrs(s);
  mutex_unlock(&slab_mutex);
  err = sysfs_slab_add(s);
  mutex_lock(&slab_mutex);
@@ -5198,6 +5200,7 @@ static ssize_t slab_attr_store(struct kobject *kobj,
  struct slab_attribute *attribute;
  struct kmem_cache *s;
  int err;
+ int i __maybe_unused;

  attribute = to_slab_attr(attr);
  s = to_slab(kobj);
@@ -5206,10 +5209,81 @@ static ssize_t slab_attr_store(struct kobject *kobj,
   return -EIO;

  err = attribute->store(s, buf, len);
+#ifdef CONFIG_MEMCG_KMEM
+ if (slab_state < FULL)
+  return err;
```

```
+ if ((err < 0) || !is_root_cache(s))
+  return err;
+
+ mutex_lock(&slab_mutex);
+ if (s->max_attr_size < len)
+  s->max_attr_size = len;
+
+ for_each_memcg_cache_index(i) {
+  struct kmem_cache *c = cache_from_memcg(s, i);
+  if (c)
+   /* return value determined by the parent cache only */
+   attribute->store(c, buf, len);
+ }
+ mutex_unlock(&slab_mutex);
+#endif
  return err;
 }

+#ifdef CONFIG_MEMCG_KMEM
+static void memcg_propagate_slab_attrs(struct kmem_cache *s)
+{
+ int i;
+ char *buffer = NULL;
+
+ if (!is_root_cache(s))
+  return;
+
+ /*
+  * This mean this cache had no attribute written. Therefore, no point
+  * in copying default values around
+  */
+ if (!s->max_attr_size)
+  return;
+
+ for (i = 0; i < ARRAY_SIZE(slab_attrs); i++) {
+  char mbuf[64];
+  char *buf;
+  struct slab_attribute *attr = to_slab_attr(slab_attrs[i]);
+
+  if (!attr || !attr->store || !attr->show)
+   continue;
+
+  /*
+   * It is really bad that we have to allocate here, so we will
+   * do it only as a fallback. If we actually allocate, though,
+   * we can just use the allocated buffer until the end.
+   *
+   * Most of the slub attributes will tend to be very small in
```

```
+   * size, but sysfs allows buffers up to a page, so they can
+   * theoretically happen.
+   */
+  if (buffer)
+   buf = buffer;
+  else if (s->max_attr_size < ARRAY_SIZE(mbuf))
+   buf = mbuf;
+  else {
+   buffer = (char *) get_zeroed_page(GFP_KERNEL);
+   if (WARN_ON(!buffer))
+    continue;
+   buf = buffer;
+  }
+
+  attr->show(s->memcg_params->root_cache, buf);
+  attr->store(s, buf, strlen(buf));
+ }
+
+ if (buffer)
+  free_page((unsigned long)buffer);
+}
+#endif
+
 static const struct sysfs_ops slab_sysfs_ops = {
  .show = slab_attr_show,
  .store = slab_attr_store,
--
1.7.11.7
```