Subject: [PATCH v4 10/14] memcg: use static branches when code not in use
Posted by Glauber Costa on Mon, 08 Oct 2012 10:06:16 GMT
View Forum Message <> Reply to Message

We can use static branches to patch the code in or out when not used.

Because the _ACTIVE bit on kmem_accounted is only set after the
increment is done, we guarantee that the root memcg will always be
selected for kmem charges until all call sites are patched (see
memcg_kmem_enabled). This guarantees that no mischarges are applied.

static branch decrement happens when the last reference count from the
kmem accounting in memcg dies. This will only happen when the charges
drop down to 0.

When that happen, we need to disable the static branch only on those
memcgs that enabled it. To achieve this, we would be forced to
complicate the code by keeping track of which memcgs were the ones
that actually enabled limits, and which ones got it from its parents.

It is a lot simpler just to do static_key_slow_inc() on every child
that is accounted.

[v4: adapted this patch to the changes in kmem_accounted]

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: Michal Hocko <mhocko@suse.cz>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Suleiman Souhlal <suleiman@google.com>
---
include/linux/memcontrol.h | 4 ++-
mm/memcontrol.c | 82 ++---
2 files changed, 80 insertions(+), 6 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 783cd78..40d658d 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -22,6 +22,7 @@
#include <linux/cgroup.h>
#include <linux/vm_event_item.h>
#include <linux/hardirq.h>
+#include <linux/jump_label.h>

struct mem_cgroup;

```
 struct page_cgroup;
@@ -401,9 +402,10 @@ struct sock;
 void sock_update_memcg(struct sock *sk);
 void sock_release_memcg(struct sock *sk);

+extern struct static_key memcg_kmem_enabled_key;
 static inline bool memcg_kmem_enabled(void)
 {
- return true;
+ return static_key_false(&memcg_kmem_enabled_key);
 }

 extern bool __memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup **memcg,
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 634c7b5..724a08b 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -344,11 +344,15 @@ struct mem_cgroup {
 /* internal only representation about the status of kmem accounting. */
 enum {
  KMEM_ACCOUNTED_ACTIVE = 0, /* accounted by this cgroup itself */
+ KMEM_ACCOUNTED_ACTIVATED, /* static key enabled. */
  KMEM_ACCOUNTED_DEAD, /* dead memcg, pending kmem charges */
 };

-/* first bit */
-#define KMEM_ACCOUNTED_MASK 0x1
+/*
+ * first two bits. We account when limit is on, but only after
+ * call sites are patched
+ */
+#define KMEM_ACCOUNTED_MASK 0x3

 #ifdef CONFIG_MEMCG_KMEM
 static void memcg_kmem_set_active(struct mem_cgroup *memcg)
@@ -361,6 +365,11 @@ static bool memcg_kmem_is_active(struct mem_cgroup *memcg)
  return test_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted);
 }

+static void memcg_kmem_set_activated(struct mem_cgroup *memcg)
+{
+ set_bit(KMEM_ACCOUNTED_ACTIVATED, &memcg->kmem_accounted);
+}
+
 static void memcg_kmem_mark_dead(struct mem_cgroup *memcg)
 {
  if (test_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted))
@@ -530,6 +539,26 @@ static void disarm_sock_keys(struct mem_cgroup *memcg)
```

```
 }
 #endif

+#ifdef CONFIG_MEMCG_KMEM
+struct static_key memcg_kmem_enabled_key;
+
+static void disarm_kmem_keys(struct mem_cgroup *memcg)
+{
+ if (memcg_kmem_is_active(memcg))
+  static_key_slow_dec(&memcg_kmem_enabled_key);
+}
+#else
+static void disarm_kmem_keys(struct mem_cgroup *memcg)
+{
+}
+#endif /* CONFIG_MEMCG_KMEM */
+
+static void disarm_static_keys(struct mem_cgroup *memcg)
+{
+ disarm_sock_keys(memcg);
+ disarm_kmem_keys(memcg);
+}
+
 static void drain_all_stock_async(struct mem_cgroup *memcg);

 static struct mem_cgroup_per_zone *
@@ -4165,6 +4194,8 @@ static int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
 {
  int ret = -EINVAL;
 #ifdef CONFIG_MEMCG_KMEM
+ bool must_inc_static_branch = false;
+
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
  /*
   * For simplicity, we won't allow this to be disabled.  It also can't
@@ -4196,7 +4227,15 @@ static int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
   if (ret)
    goto out;

-  memcg_kmem_set_active(memcg);
+  /*
+   * After this point, kmem_accounted (that we test atomically in
+   * the beginning of this conditional), is no longer 0. This
+   * guarantees only one process will set the following boolean
+   * to true. We don't need test_and_set because we're protected
+   * by the set_limit_mutex anyway.
+   */
+  memcg_kmem_set_activated(memcg);
```

```
+  must_inc_static_branch = true;
   /*
    * kmem charges can outlive the cgroup. In the case of slab
    * pages, for instance, a page contain objects from various
@@ -4209,6 +4248,27 @@ static int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
 out:
  mutex_unlock(&set_limit_mutex);
  cgroup_unlock();
+
+ /*
+  * We are by now familiar with the fact that we can't inc the static
+  * branch inside cgroup_lock. See disarm functions for details. A
+  * worker here is overkill, but also wrong: After the limit is set, we
+  * must start accounting right away. Since this operation can't fail,
+  * we can safely defer it to here - no rollback will be needed.
+  *
+  * The boolean used to control this is also safe, because
+  * KMEM_ACCOUNTED_ACTIVATED guarantees that only one process will be
+  * able to set it to true;
+  */
+ if (must_inc_static_branch) {
+  static_key_slow_inc(&memcg_kmem_enabled_key);
+  /*
+   * setting the active bit after the inc will guarantee no one
+   * starts accounting before all call sites are patched
+   */
+  memcg_kmem_set_active(memcg);
+ }
+
 #endif
  return ret;
 }
@@ -4218,8 +4278,20 @@ static void memcg_propagate_kmem(struct mem_cgroup *memcg,
 {
  memcg->kmem_accounted = parent->kmem_accounted;
 #ifdef CONFIG_MEMCG_KMEM
- if (memcg_kmem_is_active(memcg))
+ /*
+  * When that happen, we need to disable the static branch only on those
+  * memcgs that enabled it. To achieve this, we would be forced to
+  * complicate the code by keeping track of which memcgs were the ones
+  * that actually enabled limits, and which ones got it from its
+  * parents.
+  *
+  * It is a lot simpler just to do static_key_slow_inc() on every child
+  * that is accounted.
+  */
+ if (memcg_kmem_is_active(memcg)) {
```

```
   mem_cgroup_get(memcg);
+  static_key_slow_inc(&memcg_kmem_enabled_key);
+ }
 #endif
 }

@@ -5139,7 +5211,7 @@ static void free_work(struct work_struct *work)
   * to move this code around, and make sure it is outside
   * the cgroup_lock.
   */
- disarm_sock_keys(memcg);
+ disarm_static_keys(memcg);
  if (size < PAGE_SIZE)
   kfree(memcg);
  else
--
1.7.11.4
```