Subject: [PATCH 20/23] memcg: disable kmem code when not in use.
Posted by Glauber Costa on Sun, 22 Apr 2012 23:53:37 GMT
View Forum Message <> Reply to Message

We can use jump labels to patch the code in or out
when not used.

Because the assignment: memcg->kmem_accounted = true
is done after the jump labels increment, we guarantee
that the root memcg will always be selected until
all call sites are patched (see mem_cgroup_kmem_enabled).
This guarantees that no mischarges are applied.

Jump label decrement happens when the last reference
count from the memcg dies. This will only happen when
the caches are all dead.

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 |    4 +++-
 mm/memcontrol.c            |   21 +++++++++++++++++++++-
 2 files changed, 23 insertions(+), 2 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index c1c1302..25c4324 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -21,6 +21,7 @@
 #define _LINUX_MEMCONTROL_H
 #include <linux/cgroup.h>
 #include <linux/vm_event_item.h>
+#include <linux/jump_label.h>

 struct mem_cgroup;
 struct page_cgroup;
@@ -460,7 +461,8 @@ void __mem_cgroup_uncharge_kmem(size_t size);
 struct kmem_cache *
 __mem_cgroup_get_kmem_cache(struct kmem_cache *cachep, gfp_t gfp);

-#define mem_cgroup_kmem_on 1
+extern struct static_key mem_cgroup_kmem_enabled_key;
+#define mem_cgroup_kmem_on static_key_false(&mem_cgroup_kmem_enabled_key)

```
 void mem_cgroup_destroy_cache(struct kmem_cache *cachep);
 #else
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index ae61e99..547b632 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -422,6 +422,10 @@ static void mem_cgroup_put(struct mem_cgroup *memcg);
 #include <net/sock.h>
 #include <net/ip.h>

+struct static_key mem_cgroup_kmem_enabled_key;
+/* so modules can inline the checks */
+EXPORT_SYMBOL(mem_cgroup_kmem_enabled_key);
+
 static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
 static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta);
 static void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta);
@@ -468,6 +472,12 @@ void sock_release_memcg(struct sock *sk)
  }
 }

+static void disarm_static_keys(struct mem_cgroup *memcg)
+{
+ if (memcg->kmem_accounted)
+  static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
+}
+
 #ifdef CONFIG_INET
 struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
 {
@@ -847,6 +857,10 @@ static void memcg_slab_init(struct mem_cgroup *memcg)
  for (i = 0; i < MAX_KMEM_CACHE_TYPES; i++)
   rcu_assign_pointer(memcg->slabs[i], NULL);
 }
+#else
+static inline void disarm_static_keys(struct mem_cgroup *memcg)
+{
+}
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

 static void drain_all_stock_async(struct mem_cgroup *memcg);
@@ -4366,8 +4380,12 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
     *
     * But it is not worth the trouble
     */
-   if (!memcg->kmem_accounted && val != RESOURCE_MAX)
+   mutex_lock(&set_limit_mutex);
```

```
+   if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
+    static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
     memcg->kmem_accounted = true;
+   }
+   mutex_unlock(&set_limit_mutex);
   }
 #endif
   else
@@ -5349,6 +5367,7 @@ static void __mem_cgroup_put(struct mem_cgroup *memcg, int count)
 {
  if (atomic_sub_and_test(count, &memcg->refcnt)) {
   struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+  disarm_static_keys(memcg);
   __mem_cgroup_free(memcg);
   if (parent)
    mem_cgroup_put(parent);
--
1.7.7.6
```