Subject: [RFC 2/4] memcg: make it suck faster
Posted by Glauber Costa on Tue, 25 Sep 2012 08:52:51 GMT
View Forum Message <> Reply to Message

It is an accepted fact that memcg sucks. But can it suck faster? Or in a more fair statement, can it at least stop draining everyone's performance when it is not in use?

This experimental and slightly crude patch demonstrates that we can do that by using static branches to patch it out until the first memcg comes to life. There are edges to be trimmed, and I appreciate comments for direction. In particular, the events in the root are not fired, but I believe this can be done without further problems by calling a specialized event check from mem_cgroup_newpage_charge().

My goal was to have enough numbers to demonstrate the performance gain that can come from it. I tested it in a 24-way 2-socket Intel box, 24 Gb mem. I used Mel Gorman's pft test, that he used to demonstrate this problem back in the Kernel Summit. There are three kernels:

nomemcg : memcg compile disabled.
base : memcg enabled, patch not applied.
bypassed : memcg enabled, with patch applied.

| | base | bypassed |
|---|---|---|
| User | 109.12 | 105.64 |
| System | 1646.84 | 1597.98 |
| Elapsed | 229.56 | 215.76 |

| | nomemcg | bypassed |
|---|---|---|
| User | 104.35 | 105.64 |
| System | 1578.19 | 1597.98 |
| Elapsed | 212.33 | 215.76 |

So as one can see, the difference between base and nomemcg in terms of both system time and elapsed time is quite drastic, and consistent with the figures shown by Mel Gorman in the Kernel summit. This is a ~ 7 % drop in performance, just by having memcg enabled. memcg functions appear heavily in the profiles, even if all tasks lives in the root memcg.

With bypassed kernel, we drop this down to 1.5 %, which starts to fall in the acceptable range. More investigation is needed to see if we can claim that last percent back, but I believe at last part of it should be.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Michal Hocko <mhocko@suse.cz>

```
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Mel Gorman <mgorman@suse.de>
CC: Andrew Morton <akpm@linux-foundation.org>
---
 include/linux/memcontrol.h | 56 ++++++++++++++++++++++++++++++++---------
 mm/memcontrol.c            | 26 ++++++++++++++++++++------
 mm/page_cgroup.c           |  4 ++--
 3 files changed, 67 insertions(+), 19 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 6d5e212..9dabe61 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -42,6 +42,23 @@ struct mem_cgroup_reclaim_cookie {
 };

 #ifdef CONFIG_MEMCG
+extern struct static_key memcg_in_use_key;
+
+static inline bool mem_cgroup_subsys_disabled(void)
+{
+ return !!mem_cgroup_subsys.disabled;
+}
+
+static inline bool mem_cgroup_disabled(void)
+{
+ if (!static_key_false(&memcg_in_use_key))
+  return true;
+ if (mem_cgroup_subsys_disabled())
+  return true;
+ return false;
+}
+
+
 /*
  * All "charge" functions with gfp_mask should use GFP_KERNEL or
  * (gfp_mask & GFP_RECLAIM_MASK). In current implementatin, memcg doesn't
@@ -53,8 +70,18 @@ struct mem_cgroup_reclaim_cookie {
  * (Of course, if memcg does memory allocation in future, GFP_KERNEL is sane.)
  */

-extern int mem_cgroup_newpage_charge(struct page *page, struct mm_struct *mm,
+extern int __mem_cgroup_newpage_charge(struct page *page, struct mm_struct *mm,
     gfp_t gfp_mask);
+
+static inline int
+mem_cgroup_newpage_charge(struct page *page, struct mm_struct *mm,
```

```
+     gfp_t gfp_mask)
+{
+ if (mem_cgroup_disabled())
+  return 0;
+ return __mem_cgroup_newpage_charge(page, mm, gfp_mask);
+}
+
 /* for swap handling */
 extern int mem_cgroup_try_charge_swapin(struct mm_struct *mm,
   struct page *page, gfp_t mask, struct mem_cgroup **memcgp);
@@ -66,7 +93,15 @@ extern int mem_cgroup_cache_charge(struct page *page, struct
mm_struct *mm,
      gfp_t gfp_mask);

 struct lruvec *mem_cgroup_zone_lruvec(struct zone *, struct mem_cgroup *);
-struct lruvec *mem_cgroup_page_lruvec(struct page *, struct zone *);
+struct lruvec *__mem_cgroup_page_lruvec(struct page *, struct zone *);
+
+static inline struct lruvec *
+mem_cgroup_page_lruvec(struct page *page, struct zone *zone)
+{
+ if (mem_cgroup_disabled())
+  return &zone->lruvec;
+ return __mem_cgroup_page_lruvec(page, zone);
+}

 /* For coalescing uncharge for reducing memcg' overhead*/
 extern void mem_cgroup_uncharge_start(void);
@@ -129,13 +164,6 @@ extern void mem_cgroup_replace_page_cache(struct page *oldpage,
 extern int do_swap_account;
 #endif

-static inline bool mem_cgroup_disabled(void)
-{
- if (mem_cgroup_subsys.disabled)
-  return true;
- return false;
-}
-
 void __mem_cgroup_begin_update_page_stat(struct page *page, bool *locked,
      unsigned long *flags);

@@ -184,7 +212,15 @@ unsigned long mem_cgroup_soft_limit_reclaim(struct zone *zone, int
order,
      gfp_t gfp_mask,
      unsigned long *total_scanned);

-void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx);
```

```
+void __mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx);
+
+static inline void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item
idx)
+{
+ if (mem_cgroup_disabled())
+  return;
+ __mem_cgroup_count_vm_event(mm, idx);
+}
+
 #ifdef CONFIG_TRANSPARENT_HUGEPAGE
 void mem_cgroup_split_huge_fixup(struct page *head);
 #endif
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index bac398b..a2c88c4 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -472,6 +472,8 @@ struct mem_cgroup *mem_cgroup_from_css(struct cgroup_subsys_state
*s)
  return container_of(s, struct mem_cgroup, css);
 }

+struct static_key memcg_in_use_key;
+
 /* Writing them here to avoid exposing memcg's inner layout */
 #ifdef CONFIG_MEMCG_KMEM
 #include <net/sock.h>
@@ -1446,12 +1448,19 @@ static void memcg_check_events(struct mem_cgroup *memcg,
struct page *page)

 struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
 {
+ if (mem_cgroup_disabled())
+  return root_mem_cgroup;
+
  return mem_cgroup_from_css(
   cgroup_subsys_state(cont, mem_cgroup_subsys_id));
 }

 struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p)
 {
+
+ if (mem_cgroup_disabled())
+  return root_mem_cgroup;
+
  /*
   * mm_update_next_owner() may clear mm->owner to NULL
   * if it races with swapoff, page migration, etc.
```

```
@@ -1599,7 +1608,7 @@ static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)
  return (memcg == root_mem_cgroup);
 }

-void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
+void __mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
  struct mem_cgroup *memcg;

@@ -1666,15 +1675,12 @@ struct lruvec *mem_cgroup_zone_lruvec(struct zone *zone,
  * @page: the page
  * @zone: zone of the page
  */
-struct lruvec *mem_cgroup_page_lruvec(struct page *page, struct zone *zone)
+struct lruvec *__mem_cgroup_page_lruvec(struct page *page, struct zone *zone)
 {
  struct mem_cgroup_per_zone *mz;
  struct mem_cgroup *memcg;
  struct page_cgroup *pc;

- if (mem_cgroup_disabled())
-  return &zone->lruvec;
-
  pc = lookup_page_cgroup(page);
  memcg = pc->mem_cgroup;

@@ -1686,8 +1692,11 @@ struct lruvec *mem_cgroup_page_lruvec(struct page *page, struct
zone *zone)
   * Our caller holds lru_lock, and PageCgroupUsed is updated
   * under page_cgroup lock: between them, they make all uses
   * of pc->mem_cgroup safe.
+  *
+  * XXX: We can now be LRU and memcg == 0. Scanning all pages on cgroup init
+  * is too expensive. We can ultimately pay,  but prefer to do it here.
   */
- if (!PageLRU(page) && !PageCgroupUsed(pc) && memcg != root_mem_cgroup)
+ if (!PageCgroupUsed(pc) && memcg != root_mem_cgroup)
   pc->mem_cgroup = memcg = root_mem_cgroup;

  mz = page_cgroup_zoneinfo(memcg, page);
@@ -3374,7 +3383,7 @@ static int mem_cgroup_charge_common(struct page *page, struct
mm_struct *mm,
  return 0;
 }

-int mem_cgroup_newpage_charge(struct page *page,
+int __mem_cgroup_newpage_charge(struct page *page,
         struct mm_struct *mm, gfp_t gfp_mask)
```

```
 {
  if (mem_cgroup_disabled())
@@ -5703,6 +5712,8 @@ mem_cgroup_create(struct cgroup *cont)
   parent = mem_cgroup_from_cont(cont->parent);
   memcg->use_hierarchy = parent->use_hierarchy;
   memcg->oom_kill_disable = parent->oom_kill_disable;
+
+  static_key_slow_inc(&memcg_in_use_key);
  }

  if (parent && parent->use_hierarchy) {
@@ -5777,6 +5788,7 @@ static void mem_cgroup_destroy(struct cgroup *cont)

  kmem_cgroup_destroy(memcg);

+ static_key_slow_dec(&memcg_in_use_key);
  mem_cgroup_put(memcg);
 }

diff --git a/mm/page_cgroup.c b/mm/page_cgroup.c
index 5ddad0c..5699e9f 100644
--- a/mm/page_cgroup.c
+++ b/mm/page_cgroup.c
@@ -68,7 +68,7 @@ void __init page_cgroup_init_flatmem(void)

  int nid, fail;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_subsys_disabled())
   return;

  for_each_online_node(nid)  {
@@ -268,7 +268,7 @@ void __init page_cgroup_init(void)
  unsigned long pfn;
  int nid;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_subsys_disabled())
   return;

  for_each_node_state(nid, N_HIGH_MEMORY) {
--
1.7.11.4
```