Subject: [RFC}[PATCH] forced uncharge for successful rmdir.
Posted by KAMEZAWA Hiroyuki on Thu, 27 Sep 2007 07:30:35 GMT
View Forum Message <> Reply to Message

Following patch is for successful rmdir in memory cgroup.

Currently, rmdir against memory cgroup will fail if charged page caches are in cgroup.
(even if no tasks in cgroup.)

I don't like this. This patch implements "forced" uncharge against memory cgroup.
Uncharged pages will be charged again when some other cgroup/task accesses it.

I wonder that all unmapped page-caches in cgroup should be uncharged if "1" is written to control_type.

I'm grad if I can hear memory controller's policy about "rmdir" against no-task-cgroup.

Thanks,
-Kame
==
An experimental patch.

This patch adds an interface to uncharge all pages in memory cgroup if
no tasks are in it. By this, a user can remove cgroup by 'rmdir'.

To uncharge all remaing pages in cgrop, echo -n 0 to memory.control_type.
(Just for test, please advise me about better interface.
I think 'rmdir' automatically do this is an one way.)

Following is my session:
==
```
  [root@aworks kamezawa]# mkdir /opt/cgroup/group_A
  [root@aworks kamezawa]# bash
  [root@aworks kamezawa]# echo $$ > /opt/cgroup/group_A/tasks
  [root@aworks kamezawa]# cat /opt/cgroup/group_A/memory.usage_in_bytes
  122880
  [root@aworks kamezawa]# cp ./tmpfile tmpfile2
  [root@aworks kamezawa]# cat /opt/cgroup/group_A/memory.usage_in_bytes
  8597504
  [root@aworks kamezawa]# exit
  exit
  [root@aworks kamezawa]# cat /opt/cgroup/group_A/memory.usage_in_bytes
  8454144
  [root@aworks kamezawa]# cat /opt/cgroup/group_A/tasks
(*)[root@aworks kamezawa]# echo -n 0 > /opt/cgroup/group_A/memory.control_type
  [root@aworks kamezawa]# cat /opt/cgroup/group_A/memory.usage_in_bytes
  0
  [root@aworks kamezawa]# cat /opt/cgroup/group_A/tasks
```

```
  [root@aworks kamezawa]# rmdir /opt/cgroup/group_A
  [root@aworks kamezawa]# exit
==
```
In above case, a user can't remove group_A because of 8453144 bytes of
page cache. By (*), all page caches are uncharged.

uncharged pages will be charged again if some process accesses it later.
(or dropped by kswapd.)

p.s.
 extra consideration about currently mapped pages (recharge it immediately)
 will be needed ?

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

```
 mm/memcontrol.c |   93 ++++++++++++++++++++++++++++++++++++++++++++++++-------
 1 file changed, 83 insertions(+), 10 deletions(-)

Index: linux-2.6.23-rc8-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.23-rc8-mm1.orig/mm/memcontrol.c
+++ linux-2.6.23-rc8-mm1/mm/memcontrol.c
@@ -424,17 +424,80 @@ void mem_cgroup_uncharge(struct page_cgr
  if (atomic_dec_and_test(&pc->ref_cnt)) {
   page = pc->page;
   lock_page_cgroup(page);
-  mem = pc->mem_cgroup;
-  css_put(&mem->css);
-  page_assign_page_cgroup(page, NULL);
-  unlock_page_cgroup(page);
-  res_counter_uncharge(&mem->res, PAGE_SIZE);
+  pc = page_get_page_cgroup(page);
+  if (pc) {
+   mem = pc->mem_cgroup;
+   css_put(&mem->css);
+   page_assign_page_cgroup(page, NULL);
+   unlock_page_cgroup(page);
+   res_counter_uncharge(&mem->res, PAGE_SIZE);
+    spin_lock_irqsave(&mem->lru_lock, flags);
+    list_del_init(&pc->lru);
+    spin_unlock_irqrestore(&mem->lru_lock, flags);
+   kfree(pc);
+  } else
+   unlock_page_cgroup(page);
+ }
+}
```

```
+/*
+ * Uncharge pages in force. If the page is accessed again, it will be recharged by
+ * other cgroup.
+ *
+ * mem->lru_lock guarantees no-race with mem_cgroup_isolate_pages()
+ * lock_page_cgroup() -> pc = page_get_page_cgroup() guarantees no-race with
+ * mem_cgroup_uncharge().
+ */

-   spin_lock_irqsave(&mem->lru_lock, flags);
-   list_del_init(&pc->lru);
-   spin_unlock_irqrestore(&mem->lru_lock, flags);
-  kfree(pc);
+static void
+mem_cgroup_forced_uncharge_list(struct mem_cgroup *mem, struct list_head *src)
+{
+ struct page_cgroup *pc;
+ struct page *page;
+ int count = SWAP_CLUSTER_MAX;
+
+ spin_lock(&mem->lru_lock);
+ while (!list_empty(src)) {
+   pc = list_entry(src->prev, struct page_cgroup, lru);
+   /* When we uncharge page, pc->page is not cleared before
+      pc is removed from LRU. But, page->pc will be cleared. */
+   page = pc->page;
+   lock_page_cgroup(page);
+   pc = page_get_page_cgroup(page);
+   /* check race */
+   if (pc) {
+    css_put(&mem->css);
+    page_assign_page_cgroup(page, NULL);
+    unlock_page_cgroup(page);
+    res_counter_uncharge(&mem->res, PAGE_SIZE);
+    list_del_init(&pc->lru);
+    kfree(pc);
+   } else
+    unlock_page_cgroup(page);
+   if (--count == 0) {
+    spin_unlock(&mem->lru_lock);
+    cond_resched();
+    spin_lock(&mem->lru_lock);
+    count = SWAP_CLUSTER_MAX;
+   }
+ }
+ spin_unlock(&mem->lru_lock);
+}
+
```

```
+int mem_cgroup_forced_uncharge_all(struct mem_cgroup *mem)
+{
+ int ret = -EBUSY;
+ css_get(&mem->css);
+ while (!list_empty(&mem->active_list) ||
+        !list_empty(&mem->inactive_list)) {
+  if (atomic_read(&mem->css.cgroup->count) > 0)
+   goto out;
+  mem_cgroup_forced_uncharge_list(mem, &mem->active_list);
+  mem_cgroup_forced_uncharge_list(mem, &mem->inactive_list);
  }
+ ret = 0;
+out:
+ css_put(&mem->css);
+ return ret;
 }

 int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
@@ -494,7 +557,17 @@ static ssize_t mem_control_type_write(st
  if (*end != '\0')
   goto out_free;

- if (tmp <= MEM_CGROUP_TYPE_UNSPEC || tmp >= MEM_CGROUP_TYPE_MAX)
+ if (tmp == MEM_CGROUP_TYPE_UNSPEC) {
+  if (atomic_read(&mem->css.cgroup->count) == 0)  /* uncharge all */
+   ret = mem_cgroup_forced_uncharge_all(mem);
+  else
+   ret = -EBUSY;
+  if (!ret)
+   ret = nbytes;
+  goto out_free;
+ }
+
+ if (tmp < MEM_CGROUP_TYPE_UNSPEC || tmp >= MEM_CGROUP_TYPE_MAX)
   goto out_free;

  mem->control_type = tmp;
```