Subject: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Fri, 06 Jun 2008 01:52:35 GMT
View Forum Message <> Reply to Message

Move Usage at Task Move (just an experimantal for discussion)
I tested this but don't think bug-free.

In current memcg, when task moves to a new cg, the usage remains in the old cg.
This is considered to be not good.

This is a trial to move "usage" from old cg to new cg at task move.
Finally, you'll see the problems we have to handle are failure and rollback.

This one's Basic algorithm is

0. can_attach() is called.
1. count movable pages by scanning page table. isolate all pages from LRU.
2. try to create enough room in new memory cgroup
3. start moving page accouing
4. putback pages to LRU.
5. can_attach() for other cgroups are called.

A case study.

group_A -> limit=1G, task_X's usage= 800M.
group_B -> limit=1G, usage=500M.

For moving task_X from group_A to group_B.
- group_B should be reclaimed or have enough room.

While moving task_X from group_A to group_B.
- group_B's memory usage can be changed
- group_A's memory usage can be changed

We accounts the resouce based on pages. Then, we can't move all resource
usage at once.

If group_B has no more room when we've moved 700M of task_X to group_B,
we have to move 700M of task_X back to group_A. So I implemented roll-back.
But other process may use up group_A's available resource at that point.

For avoiding that, preserve 800M in group_B before moving task_X means that
task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)

This patch uses Best-Effort rollback. Failure in rollback is ignored and
the usage is just leaked.

Roll-back can happen when

(a) in phase 3. cannot move a page to new cgroup because of limit.
(b) in phase 5. other cgourp subsys returns error in can_attach().

Rollback (a) is handled in memcg, but there is a chance for leak of accounting
at rollback. To handle rollback (b), attach_rollback() is added to cgroup_ops.
(If memcg is the last subysys, handling (b) is not necessary.)

I wonder what kind of technique can we use to avoid complicated situation....

For avoiding complicated rollbacks,
I think of following ways of policy for task moving (you can add here.)

1. Before moving usage, reserve usage in the new cgroup and old cgroup.
   Pros.
    - rollback will be very easy.
   Cons.
    - A task will use twice of its own usage virtaually for a while.
    - some amount of cpu time will be necessary to move _Big_ apps.
    - It's difficut to move _Big_ apps to small memcg.
    - we have to add "special case" handling.

2. Don't move any usage at task move. (current implementation.)
   Pros.
    - no complication in the code.
   Cons.
    - A task's usage is chareged to wrong cgroup.
    - Not sure, but I believe the users don't want this.

3. Use Lazy Manner
    When the task moves, we can mark the pages used by it as
    "Wrong Charge, Should be dropped", and add them some penalty in the LRU.
   Pros.
    - no complicated ones.
    - the pages will be gradually moved at memory pressure.
   Cons.
    - A task's usage can exceed the limit for a while.
    - can't handle mlocked() memory in proper way.

4. Allow Half-moved state and abandon rollback.
   Pros.
    - no complicated ones in the code.
   Cons.
    - the users will be in chaos.

After writing this patch, for me, "3" is attractive. now.
(or using Lazy manner and allow moving of usage instead of freeing it.)

One reasone is that I think a typical usage of memory controller is
fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.

How about you ?

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

---
 include/linux/cgroup.h | 2
 kernel/cgroup.c | 9 +-
 mm/memcontrol.c | 218 ++---
 3 files changed, 217 insertions(+), 12 deletions(-)

Index: temp-2.6.26-rc2-mm1/mm/memcontrol.c
===
--- temp-2.6.26-rc2-mm1.orig/mm/memcontrol.c
+++ temp-2.6.26-rc2-mm1/mm/memcontrol.c
@@ -32,6 +32,8 @@
 #include <linux/fs.h>
 #include <linux/seq_file.h>
 #include <linux/vmalloc.h>
+#include <linux/migrate.h>
+#include <linux/hugetlb.h>

 #include <asm/uaccess.h>

@@ -285,6 +287,34 @@ static void unlock_page_cgroup(struct pa
  bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
 }

+/*
+ * returns page_cgroup()'s mem_cgroup and its charge type.
+ * If no page_cgroup, return NULL.
+ */
+
+struct mem_cgroup *page_cgroup_get_info(struct page *page,
+ enum charge_type *ctype, int getref)
+{
+ struct mem_cgroup *mem = NULL;
+ struct page_cgroup *pc;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc) {
+ mem = pc->mem_cgroup;
+ if (getref)
+   css_get(&mem->css);

```
+
+  if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
+   *ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
+  else
+   *ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
+ }
+ unlock_page_cgroup(page);
+
+ return mem;
+}
+
 static void __mem_cgroup_remove_list(struct mem_cgroup_per_zone *mz,
    struct page_cgroup *pc)
 {
@@ -689,7 +719,6 @@ __mem_cgroup_uncharge_common(struct page
  pc = page_get_page_cgroup(page);
  if (unlikely(!pc))
   goto unlock;
-
  VM_BUG_ON(pc->page != page);

  if ((ctype == MEM_CGROUP_CHARGE_TYPE_MAPPED)
@@ -732,7 +761,6 @@ void mem_cgroup_uncharge_cache_page(stru
  */
 int mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
 {
- struct page_cgroup *pc;
  struct mem_cgroup *mem = NULL;
  enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
  int ret = 0;
@@ -740,15 +768,8 @@ int mem_cgroup_prepare_migration(struct
  if (mem_cgroup_subsys.disabled)
   return 0;

- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (pc) {
-  mem = pc->mem_cgroup;
-  css_get(&mem->css);
-  if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
-   ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
- }
- unlock_page_cgroup(page);
+ mem = page_cgroup_get_info(page, &ctype, 1);
+
  if (mem) {
   ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
    ctype, mem);
```

```
@@ -766,6 +787,179 @@ void mem_cgroup_end_migration(struct pag
      MEM_CGROUP_CHARGE_TYPE_FORCE);
 }

+static int
+mem_cgroup_recharge_private(struct page *page, struct mem_cgroup *memcg)
+{
+ int ret;
+
+ if (page_count(page) != 2
+     || page_mapcount(page) != 1
+     || !PageAnon(page))
+  return 0;
+
+
+ __mem_cgroup_uncharge_common(page, MEM_CGROUP_CHARGE_TYPE_FORCE);
+
+ /*
+  * Here, this page is not assigned to any cgroup
+  * reassign this to....
+  */
+ /* recharge to new group */
+ ret = mem_cgroup_charge_common(page, NULL, GFP_KERNEL,
+    MEM_CGROUP_CHARGE_TYPE_MAPPED, memcg);
+
+ return ret;
+}
+
+struct recharge_info {
+ struct list_head list;
+ struct vm_area_struct *vma;
+ int count;
+};
+
+static int __recharge_get_page_range(pmd_t *pmd, unsigned long addr,
+  unsigned long end, void *private)
+{
+ struct recharge_info *info = private;
+ struct vm_area_struct *vma = info->vma;
+ pte_t *pte, ptent;
+ spinlock_t *ptl;
+ struct page *page;
+
+ pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ for (; addr != end; addr += PAGE_SIZE, pte++) {
+  ptent = *pte;
+  if (!pte_present(ptent))
+   continue;
```

```
+  page = vm_normal_page(vma, addr, ptent);
+  if (!page || !PageAnon(page) || page_mapcount(page) > 1)
+   continue;
+  get_page(page);
+  if (!isolate_lru_page(page, &info->list))
+   info->count++;
+  put_page(page);
+ }
+ pte_unmap_unlock(pte - 1, ptl);
+ cond_resched();
+ return 0;
+}
+
+struct mm_walk recharge_walk = {
+ .pmd_entry = __recharge_get_page_range,
+};
+
+
+int mem_cgroup_recharge_task(struct mem_cgroup *newcg,
+    struct task_struct *task)
+{
+ struct mm_struct *mm;
+ struct vm_area_struct *vma;
+ struct mem_cgroup *oldcg;
+ struct page *page, *page2;
+ LIST_HEAD(moved);
+ struct recharge_info info;
+ int rc, necessary;
+
+ if (!newcg)
+  return 0;
+
+ mm = get_task_mm(task);
+ if (!mm)
+  return 0;
+
+ oldcg = mem_cgroup_from_task(task);
+
+ INIT_LIST_HEAD(&info.list);
+ info.count = 0;
+
+ down_read(&mm->mmap_sem);
+ for (vma = mm->mmap; vma; vma = vma->vm_next) {
+  /* We just recharge Private pages. */
+  if (is_vm_hugetlb_page(vma) ||
+   vma->vm_flags & (VM_SHARED | VM_MAYSHARE))
+   continue;
+  info.vma = vma;
```

```
+  walk_page_range(mm, vma->vm_start, vma->vm_end,
+    &recharge_walk, &info);
+ }
+ up_read(&mm->mmap_sem);
+ mmput(mm);
+
+
+ /* create enough room before move */
+ necessary = info.count * PAGE_SIZE;
+
+ do {
+  spin_lock(&newcg->res.lock);
+  if (newcg->res.limit > necessary)
+   rc = -ENOMEM;
+  if (newcg->res.usage + necessary > newcg->res.limit)
+   rc = 1;
+  else
+   rc = 0;
+  spin_unlock(&newcg->res.lock);
+
+  if (rc == -ENOMEM)
+   break;
+
+  if (rc) { /* need to reclaim some ? */
+   int progress;
+   progress = try_to_free_mem_cgroup_pages(newcg,
+     GFP_KERNEL);
+   rc = -ENOMEM;
+   if (!progress)
+    break;
+  } else
+   break;
+  cond_resched();
+ } while (1);
+
+ if (rc)
+  goto end;
+
+ list_for_each_entry_safe(page, page2, &info.list, lru) {
+  cond_resched();
+  /* Here this page is the target of rollback */
+  list_move(&page->lru, &moved);
+  rc = mem_cgroup_recharge_private(page, newcg);
+
+  if (rc)
+   goto rollback;;
+ }
+end:
```

```
+ putback_lru_pages(&info.list);
+ putback_lru_pages(&moved);
+ return rc;
+
+rollback:
+ /* at faiulre Move back all to oldcg */
+ list_for_each_entry_safe(page, page2, &moved, lru) {
+  cond_resched();
+  mem_cgroup_recharge_private(page, oldcg);
+  /* ignore this failure intentionally. this will cause that
+     the page is not charged to anywhere. */
+ }
+ goto end;
+}
+
+int mem_cgroup_can_attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
+   struct task_struct *tsk)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ return mem_cgroup_recharge_task(memcg, tsk);
+}
+
+void mem_cgroup_attach_rollback(struct cgroup_subsys *ss,
+   struct task_struct *tsk)
+{
+ struct mem_cgroup *memcg;
+
+ rcu_read_lock();
+ memcg = mem_cgroup_from_task(tsk);
+ rcu_read_unlock();
+ mem_cgroup_recharge_task(memcg, tsk);
+}
+
 /*
  * A call to try to shrink memory usage under specified resource controller.
  * This is typically used for page reclaiming for shmem for reducing side
@@ -1150,6 +1344,8 @@ struct cgroup_subsys mem_cgroup_subsys =
  .pre_destroy = mem_cgroup_pre_destroy,
  .destroy = mem_cgroup_destroy,
  .populate = mem_cgroup_populate,
+ .can_attach = mem_cgroup_can_attach,
+ .attach_rollback = mem_cgroup_attach_rollback,
  .attach = mem_cgroup_move_task,
  .early_init = 0,
 };
Index: temp-2.6.26-rc2-mm1/include/linux/cgroup.h
===================================================================
--- temp-2.6.26-rc2-mm1.orig/include/linux/cgroup.h
```

```
+++ temp-2.6.26-rc2-mm1/include/linux/cgroup.h
@@ -299,6 +299,8 @@ struct cgroup_subsys {
      struct cgroup *cgrp, struct task_struct *tsk);
  void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
    struct cgroup *old_cgrp, struct task_struct *tsk);
+ void (*attach_rollback)(struct cgroup_subsys *ss,
+    struct task_struct *tsk);
  void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
  void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
  int (*populate)(struct cgroup_subsys *ss,
Index: temp-2.6.26-rc2-mm1/kernel/cgroup.c
===================================================================
--- temp-2.6.26-rc2-mm1.orig/kernel/cgroup.c
+++ temp-2.6.26-rc2-mm1/kernel/cgroup.c
@@ -1241,7 +1241,7 @@ int cgroup_attach_task(struct cgroup *cg
   if (ss->can_attach) {
    retval = ss->can_attach(ss, cgrp, tsk);
    if (retval)
-    return retval;
+    goto rollback;
   }
  }

@@ -1278,6 +1278,13 @@ int cgroup_attach_task(struct cgroup *cg
  synchronize_rcu();
  put_css_set(cg);
  return 0;
+
+rollback:
+ for_each_subsys(root, ss) {
+  if (ss->attach_rollback)
+   ss->attach_rollback(ss, tsk);
+ }
+ return retval;
 }

 /*
```