
Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [5/6]
memory cgroup and migration fix
Posted by [Balbir Singh](#) on Tue, 09 Oct 2007 16:26:02 GMT
[View Forum Message](#) <> [Reply to Message](#)

KAMEZAWA Hiroyuki wrote:

> While using memory control cgroup, page-migration under it works as following.

> ==

> 1. uncharge all refs at try to unmap.

> 2. charge regs again remove_migration_ptes()

> ==

> This is simple but has following problems.

> ==

> The page is uncharged and charged back again if *mapped*.

> - This means that cgroup before migration can be different from one after

> migration

> - If page is not mapped but charged as page cache, charge is just ignored
> (because not mapped, it will not be uncharged before migration)

> This is memory leak.

> ==

> This patch tries to keep memory cgroup at page migration by increasing
> one refcnt during it. 3 functions are added.

>

> mem_cgroup_prepare_migration() --- increase refcnt of page->page_cgroup

> mem_cgroup_end_migration() --- decrease refcnt of page->page_cgroup

> mem_cgroup_page_migration() --- copy page->page_cgroup from old page to
> new page.

>

> During migration

> - old page is under PG_locked.

> - new page is under PG_locked, too.

> - both old page and new page are not on LRU.

>

> These 3 facts guarantees page_cgroup() migration has no race, I think.

>

> Tested and worked well in x86_64/fake-NUMA box.

>

> Changelog v1 -> v2:

> - reflected comments.

> - divided a patch to !PageLRU patch and migration patch.

>

>

>

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

>

> include/linux/memcontrol.h | 19 ++++++

> mm/memcontrol.c | 43 ++++++

> mm/migrate.c | 14 +++++-

```

> 3 files changed, 73 insertions(+), 3 deletions(-)
>
> Index: devel-2.6.23-rc8-mm2/mm/migrate.c
> =====
> --- devel-2.6.23-rc8-mm2.orig/mm/migrate.c
> +++ devel-2.6.23-rc8-mm2/mm/migrate.c
> @@ -598,9 +598,10 @@ static int move_to_new_page(struct page
>  else
>  rc = fallback_migrate_page(mapping, newpage, page);
>
> - if (!rc)
> + if (!rc) {
> + mem_cgroup_page_migration(page, newpage);
>  remove_migration_ptes(page, newpage);
> - else
> + } else
>  newpage->mapping = NULL;
>
>  unlock_page(newpage);
> @@ -619,6 +620,7 @@ static int unmap_and_move(new_page_t get
>  int *result = NULL;
>  struct page *newpage = get_new_page(page, private, &result);
>  int rcu_locked = 0;
> + int charge = 0;
>
>  if (!newpage)
>  return -ENOMEM;
> @@ -660,14 +662,20 @@ static int unmap_and_move(new_page_t get
>  */
>  if (!page->mapping)
>  goto rcu_unlock;
> +
> + charge = mem_cgroup_prepare_migration(page);
>  /* Establish migration ptes or remove ptes */
>  try_to_unmap(page, 1);
>
>  if (!page_mapped(page))
>  rc = move_to_new_page(newpage, page);
>
> - if (rc)
> + if (rc) {
>  remove_migration_ptes(page, page);
> + if (charge)
> + mem_cgroup_end_migration(page);
> + } else if (charge)
> + mem_cgroup_end_migration(newpage);
>  rcu_unlock:
>  if (rcu_locked)

```

```

> rcu_read_unlock();
> Index: devel-2.6.23-rc8-mm2/include/linux/memcontrol.h
> =====
> --- devel-2.6.23-rc8-mm2.orig/include/linux/memcontrol.h
> +++ devel-2.6.23-rc8-mm2/include/linux/memcontrol.h
> @@ -56,6 +56,10 @@ static inline void mem_cgroup_uncharge_p
> mem_cgroup_uncharge(page_get_page_cgroup(page));
> }
>
> +extern int mem_cgroup_prepare_migration(struct page *page);
> +extern void mem_cgroup_end_migration(struct page *page);
> +extern void mem_cgroup_page_migration(struct page *page, struct page *newpage);
> +
> #else /* CONFIG_CGROUP_MEM_CONT */
> static inline void mm_init_cgroup(struct mm_struct *mm,
>     struct task_struct *p)
> @@ -107,6 +111,21 @@ static inline struct mem_cgroup *mm_cgrou
> return NULL;
> }
>
> +static inline int mem_cgroup_prepare_migration(struct page *page)
> +{
> + return 0;
> +}
> +
> +static inline void mem_cgroup_end_migration(struct page *page)
> +{
> +}
> +
> +static inline void
> +mem_cgroup_page_migration(struct page *page, struct page *newpage);
> +{
> +}
> +
> +
> #endif /* CONFIG_CGROUP_MEM_CONT */
>
> #endif /* _LINUX_MEMCONTROL_H */
> Index: devel-2.6.23-rc8-mm2/mm/memcontrol.c
> =====
> --- devel-2.6.23-rc8-mm2.orig/mm/memcontrol.c
> +++ devel-2.6.23-rc8-mm2/mm/memcontrol.c
> @@ -463,6 +463,49 @@ void mem_cgroup_uncharge(struct page_cgr
> }
> }
> }
> +/*
> + * Returns non-zero if a page (under migration) has valid page_cgroup member.

```

```

> + * Refcnt of page_cgroup is incremented.
> + */
> +
> +int mem_cgroup_prepare_migration(struct page *page)
> +{
> + struct page_cgroup *pc;
> + int ret = 0;
> + lock_page_cgroup(page);
> + pc = page_get_page_cgroup(page);
> + if (pc && atomic_inc_not_zero(&pc->ref_cnt))
> + ret = 1;
> + unlock_page_cgroup(page);
> + return ret;
> +}
> +
> +void mem_cgroup_end_migration(struct page *page)
> +{
> + struct page_cgroup *pc = page_get_page_cgroup(page);
> + mem_cgroup_uncharge(pc);
> +}
> +/*
> + * We know both *page* and *newpage* are now not-on-LRU and Pg_locked.
> + * And no race with uncharge() routines because page_cgroup for *page*
> + * has extra one reference by mem_cgroup_prepare_migration.
> + */
> +
> +void mem_cgroup_page_migration(struct page *page, struct page *newpage)
> +{
> + struct page_cgroup *pc;
> +retry:
> + pc = page_get_page_cgroup(page);
> + if (!pc)
> + return;
> + if (clear_page_cgroup(page, pc) != pc)
> + goto retry;
> + pc->page = newpage;
> + lock_page_cgroup(newpage);
> + page_assign_page_cgroup(newpage, pc);
> + unlock_page_cgroup(newpage);
> + return;
> +}
>
> int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
> {
>

```

Looks good to me

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
