Subject: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [0/1]
Posted by KAMEZAWA Hiroyuki on Tue, 02 Oct 2007 09:30:31 GMT
View Forum Message <> Reply to Message

Current implementation of memory cgroup controller does following in migration.

1. uncharge when unmapped.
2. charge again when remapped.

Consider migrate a page from OLD to NEW.

In following case, memory (for page_cgroup) will leak.

1. charge OLD page as page-cache. (charge = 1
2. A process mmap OLD page. (chage + 1 = 2)
3. A process migrates it.
   try_to_unmap(OLD) (charge - 1 = 1)
   replace OLD with NEW
   remove_migration_pte(NEW) (New is newly charged.)
   discard OLD page. (page_cgroup for OLD page is not reclaimed.)

patch is in the next mail.

Test Log on 2.6.18-rc8-mm2.
==
mount cgroup and create group_A group_B
[root@drpq kamezawa]# mount -t cgroup none /opt/mem_control/ -o memory
[root@drpq kamezawa]# mkdir /opt/mem_control/group_A/
[root@drpq kamezawa]# mkdir /opt/mem_control/group_B/
[root@drpq kamezawa]# bash
[root@drpq kamezawa]# echo $$ > /opt/mem_control/group_A/tasks
[root@drpq kamezawa]# cat /opt/mem_control/group_A/memory.usage_in_bytes
475136
[root@drpq kamezawa]# grep size-64 /proc/slabinfo
size-64(DMA)   0   0   64 240   1 : tunables 120   60   8 : slabdata   0   0   0
size-64   30425 30960   64 240   1 : tunables 120   60   8 : slabdata   129   129   12

charge file cache 512Mfile to groupA
[root@drpq kamezawa]# cat 512Mfile > /dev/null
[root@drpq kamezawa]# cat /opt/mem_control/group_A/memory.usage_in_bytes
539525120

for test, try drop_caches. drop_cache works well and chage decreased.
[root@drpq kamezawa]# echo 3 > /proc/sys/vm/drop_caches
[root@drpq kamezawa]# cat /opt/mem_control/group_A/memory.usage_in_bytes
983040

```
# chage file cache 512Mfile again.
[root@drpq kamezawa]# taskset 01 cat 512Mfile > /dev/null
[root@drpq kamezawa]# exit
exit
[root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
539738112
0
[root@drpq kamezawa]# bash
#enter group B
[root@drpq kamezawa]# echo $$ > /opt/mem_control/group_B/tasks
[root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
539738112
557056
[root@drpq kamezawa]#  grep size-64 /proc/slabinfo
size-64(DMA)          0      0     64  240    1 : tunables  120   60    8 : slabdata      0      0      0
size-64           48263  59760     64  240    1 : tunables  120   60    8 : slabdata    249    249     12
# migrate_test mmaps 512Mfile and call system call move_pages(). and sleep.
[root@drpq kamezawa]# ./migrate_test 512Mfile 1 &
[1] 4108
#At the end of migration,
[root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
539738112
537706496

#Wow, charge is twice ;)
[root@drpq kamezawa]#  grep size-64 /proc/slabinfo
size-64(DMA)          0      0     64  240    1 : tunables  120   60    8 : slabdata      0      0      0
size-64           81180  92400     64  240    1 : tunables  120   60    8 : slabdata    385    385     12

#Kill migrate_test, because 512Mfile is unmapped, charge in group_B is dropped.
[root@drpq kamezawa]# kill %1
[root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
536936448
1458176
[1]+  Terminated              ./migrate_test 512Mfile 1

#Try drop caches again
[root@drpq kamezawa]# echo 3 > /proc/sys/vm/drop_caches
[root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
536920064
1097728
#no change because charge in group_A is leaked.....

[root@drpq kamezawa]#  grep size-64 /proc/slabinfo
size-64(DMA)          0      0     64  240    1 : tunables  120   60    8 : slabdata      0      0      0
size-64           48137  60720     64  240    1 : tunables  120   60    8 : slabdata    253    253    210
[root@drpq kamezawa]#
```

==

-Kame

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

---

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [1/1] fix pag
Posted by KAMEZAWA Hiroyuki on Tue, 02 Oct 2007 09:33:06 GMT
View Forum Message <> Reply to Message

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 chaged back again if *mapped*.
- This means that cgroup before migraion can be different from one after migraion
- 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 is bad.
And migration can migrate *not mapped* pages in future by migration-by-kernel driven by memory-unplug and defragment-by-migration at el.

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.

Obviously, mem_cgroup_isolate_pages() and this page migration, which copies page_cgroup from old page to new page, has race.

There seem to be 3 ways for avoiding this race.
A. take mem_group->lock while mem_cgroup_page_migration().
B. isolate pc from mem_cgroup's LRU when we isolate page from zone's LRU.
C. ignore non-LRU page at mem_cgroup_isolate_pages().

This patch uses method (C.) and modifes mem_cgroup_isolate_pages() igonres !PageLRU pages.

Tested and worked well in ia64/NUMA box.

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

---
 include/linux/memcontrol.h | 22 +++++++++++++++
 mm/memcontrol.c | 62 ++---
 mm/migrate.c | 13 +++++++--
 3 files changed, 90 insertions(+), 7 deletions(-)

Index: linux-2.6.23-rc8-mm2/include/linux/memcontrol.h
===
--- linux-2.6.23-rc8-mm2.orig/include/linux/memcontrol.h
+++ linux-2.6.23-rc8-mm2/include/linux/memcontrol.h
@@ -43,8 +43,14 @@ extern unsigned long mem_cgroup_isolate_
 	struct mem_cgroup *mem_cont,
 	int active);
 extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem, gfp_t gfp_mask);
+
 extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
 	gfp_t gfp_mask);
+/* For handling page migration in proper way */
+extern void mem_cgroup_prepare_migration(struct page *page);
+extern void mem_cgroup_end_migration(struct page *page);
+extern void mem_cgroup_page_migration(struct page *newpage, struct page *page);
+

 static inline struct mem_cgroup *mm_cgroup(const struct mm_struct *mm)
 {
@@ -107,6 +113,22 @@ static inline struct mem_cgroup *mm_cgro
  return NULL;
 }

+/* For page migration */
+static inline void mem_cgroup_prepare_migration(struct page *page)
+{
+ return;
+}
+
+static inline void mem_cgroup_end_migration(struct mem_cgroup *cgroup)
+{
+ return;
+}
+
+static inline void

```
+mem_cgroup_page_migration(struct page *newpage, struct page *page)
+{
+ return;
+}
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
Index: linux-2.6.23-rc8-mm2/mm/memcontrol.c
===================================================================
--- linux-2.6.23-rc8-mm2.orig/mm/memcontrol.c
+++ linux-2.6.23-rc8-mm2/mm/memcontrol.c
@@ -198,7 +198,7 @@ unsigned long mem_cgroup_isolate_pages(u
  unsigned long scan;
  LIST_HEAD(pc_list);
  struct list_head *src;
- struct page_cgroup *pc;
+ struct page_cgroup *pc, *tmp;

  if (active)
   src = &mem_cont->active_list;
@@ -206,8 +206,10 @@ unsigned long mem_cgroup_isolate_pages(u
   src = &mem_cont->inactive_list;

  spin_lock(&mem_cont->lru_lock);
- for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
-  pc = list_entry(src->prev, struct page_cgroup, lru);
+ scan = 0;
+ list_for_each_entry_safe_reverse(pc, tmp, src, lru) {
+  if (scan++ >= nr_to_scan)
+   break;
   page = pc->page;
   VM_BUG_ON(!pc);

@@ -225,9 +227,14 @@ unsigned long mem_cgroup_isolate_pages(u
    /*
     * Reclaim, per zone
     * TODO: make the active/inactive lists per zone
+    * !PageLRU page can be found under us while migration.
+    * just ignore it.
     */
-  if (page_zone(page) != z)
+  if (page_zone(page) != z || !PageLRU(page)) {
+   /* Skip this */
+   /* Don't decrease scan here for avoiding dead lock */
    continue;
+  }

   /*
```

```
    * Check if the meta page went away from under us
@@ -417,8 +424,14 @@ void mem_cgroup_uncharge(struct page_cgr
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+retry:
   page = pc->page;
   lock_page_cgroup(page);
+  /* migration occur ? */
+  if (page_get_page_cgroup(page) != pc) {
+   unlock_page_cgroup(page);
+   goto retry;
+  }
   mem = pc->mem_cgroup;
   css_put(&mem->css);
   page_assign_page_cgroup(page, NULL);
@@ -432,6 +445,47 @@ void mem_cgroup_uncharge(struct page_cgr
  }
 }

+void mem_cgroup_prepare_migration(struct page *page)
+{
+ struct page_cgroup *pc;
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc)
+  atomic_inc(&pc->ref_cnt);
+ unlock_page_cgroup(page);
+ return;
+}
+
+void mem_cgroup_end_migration(struct page *page)
+{
+ struct page_cgroup *pc = page_get_page_cgroup(page);
+ mem_cgroup_uncharge(pc);
+}
+
+void mem_cgroup_page_migration(struct page *newpage, struct page *page)
+{
+ struct page_cgroup *pc;
+ /*
+ * We already keep one reference to paage->cgroup.
+ * and both pages are guaranteed to be locked under page migration.
+ */
+ lock_page_cgroup(page);
+ lock_page_cgroup(newpage);
+ pc = page_get_page_cgroup(page);
+ if (!pc)
```

```
+  goto unlock_out;
+ page_assign_page_cgroup(page, NULL);
+ pc->page = newpage;
+ page_assign_page_cgroup(newpage, pc);
+
+unlock_out:
+ unlock_page_cgroup(newpage);
+ unlock_page_cgroup(page);
+ return;
+}
+
+
+
 int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
 {
  *tmp = memparse(buf, &buf);
Index: linux-2.6.23-rc8-mm2/mm/migrate.c
===================================================================
--- linux-2.6.23-rc8-mm2.orig/mm/migrate.c
+++ linux-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(newpage, page);
   remove_migration_ptes(page, newpage);
- else
+ } else
   newpage->mapping = NULL;

  unlock_page(newpage);
@@ -651,6 +652,8 @@ static int unmap_and_move(new_page_t get
   rcu_read_lock();
   rcu_locked = 1;
  }
+ mem_cgroup_prepare_migration(page);
+
  /*
   * This is a corner case handling.
   * When a new swap-cache is read into, it is linked to LRU
@@ -666,8 +669,12 @@ static int unmap_and_move(new_page_t get
  if (!page_mapped(page))
   rc = move_to_new_page(newpage, page);

- if (rc)
+ if (rc) {
```

```
   remove_migration_ptes(page, page);
+  mem_cgroup_end_migration(page);
+ } else
+  mem_cgroup_end_migration(newpage);
+
 rcu_unlock:
  if (rcu_locked)
   rcu_read_unlock();
```

Subject: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [2/1] additional
Posted by KAMEZAWA Hiroyuki on Tue, 02 Oct 2007 10:12:17 GMT
View Forum Message <> Reply to Message

The patch I sent needs following fix, sorry.
Anyway, I'll repost good-version with reflected comments again.

Thanks,
 -Kame
=
fix-page-migration-under-memory-controller patch needs this fix..sorry.

 - We should uncharge page by mem_cgroup_end_migration() only if
   page is not charged by mem_cgroup_prepare_migration().
 - move mem_cgroup_prepare_migration() after goto:

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

```
Index: linux-2.6.23-rc8-mm2/mm/memcontrol.c
===================================================================
--- linux-2.6.23-rc8-mm2.orig/mm/memcontrol.c
+++ linux-2.6.23-rc8-mm2/mm/memcontrol.c
@@ -445,7 +445,7 @@ retry:
  }
 }

-void mem_cgroup_prepare_migration(struct page *page)
+int mem_cgroup_prepare_migration(struct page *page)
 {
  struct page_cgroup *pc;
  lock_page_cgroup(page);
@@ -453,7 +453,7 @@ void mem_cgroup_prepare_migration(struct
```

```
  if (pc)
   atomic_inc(&pc->ref_cnt);
  unlock_page_cgroup(page);
- return;
+ return (pc != NULL);
 }

 void mem_cgroup_end_migration(struct page *page)
Index: linux-2.6.23-rc8-mm2/include/linux/memcontrol.h
===================================================================
--- linux-2.6.23-rc8-mm2.orig/include/linux/memcontrol.h
+++ linux-2.6.23-rc8-mm2/include/linux/memcontrol.h
@@ -47,7 +47,7 @@ extern void mem_cgroup_out_of_memory(str
 extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
      gfp_t gfp_mask);
 /* For handling page migration in proper way */
-extern void mem_cgroup_prepare_migration(struct page *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 *newpage, struct page *page);

@@ -114,9 +114,9 @@ static inline struct mem_cgroup *mm_cgro
 }

 /* For page migration */
-static inline void mem_cgroup_prepare_migration(struct page *page)
+static inline int mem_cgroup_prepare_migration(struct page *page)
 {
- return;
+ return 0;
 }

 static inline void mem_cgroup_end_migration(struct mem_cgroup *cgroup)
Index: linux-2.6.23-rc8-mm2/mm/migrate.c
===================================================================
--- linux-2.6.23-rc8-mm2.orig/mm/migrate.c
+++ linux-2.6.23-rc8-mm2/mm/migrate.c
@@ -620,6 +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;
@@ -652,8 +653,6 @@ static int unmap_and_move(new_page_t get
   rcu_read_lock();
   rcu_locked = 1;
```

```
  }
- mem_cgroup_prepare_migration(page);
-
  /*
   * This is a corner case handling.
   * When a new swap-cache is read into, it is linked to LRU
@@ -663,6 +662,8 @@ 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);

@@ -671,8 +672,9 @@ static int unmap_and_move(new_page_t get

  if (rc) {
   remove_migration_ptes(page, page);
-  mem_cgroup_end_migration(page);
- } else
+  if (charge)
+   mem_cgroup_end_migration(page);
+ } else if (charge)
   mem_cgroup_end_migration(newpage);

 rcu_unlock:
```

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [0/1]

Posted by Balbir Singh on Tue, 02 Oct 2007 13:34:15 GMT

View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Current implementation of memory cgroup controller does following in migration.
>
> 1. uncharge when unmapped.
> 2. charge again when remapped.
>
> Consider migrate a page from OLD to NEW.
>
> In following case, memory (for page_cgroup) will leak.

>
> 1. charge OLD page as page-cache. (charge = 1
> 2. A process mmap OLD page. (chage + 1 = 2)
> 3. A process migrates it.
> try_to_unmap(OLD) (charge - 1 = 1)
> replace OLD with NEW
> remove_migration_pte(NEW) (New is newly charged.)
> discard OLD page. (page_cgroup for OLD page is not reclaimed.)
>

Interesting test scenario, I'll try and reproduce the problem here.
Why does discard OLD page not reclaim page_cgroup?

> patch is in the next mail.
>

Thanks

> Test Log on 2.6.18-rc8-mm2.
> ==
> # mount cgroup and create group_A group_B
> [root@drpq kamezawa]# mount -t cgroup none /opt/mem_control/ -o memory
> [root@drpq kamezawa]# mkdir /opt/mem_control/group_A/
> [root@drpq kamezawa]# mkdir /opt/mem_control/group_B/
> [root@drpq kamezawa]# bash
> [root@drpq kamezawa]# echo $$ > /opt/mem_control/group_A/tasks
> [root@drpq kamezawa]# cat /opt/mem_control/group_A/memory.usage_in_bytes
> 475136
> [root@drpq kamezawa]# grep size-64 /proc/slabinfo
> size-64(DMA) 0 0 64 240 1 : tunables 120 60 8 : slabdata 0 0 0
> size-64 30425 30960 64 240 1 : tunables 120 60 8 : slabdata 129 129 12
>
> # charge file cache 512Mfile to groupA
> [root@drpq kamezawa]# cat 512Mfile > /dev/null
> [root@drpq kamezawa]# cat /opt/mem_control/group_A/memory.usage_in_bytes
> 539525120
>
> # for test, try drop_caches. drop_cache works well and chage decreased.
> [root@drpq kamezawa]# echo 3 > /proc/sys/vm/drop_caches
> [root@drpq kamezawa]# cat /opt/mem_control/group_A/memory.usage_in_bytes
> 983040
>
> # chage file cache 512Mfile again.
> [root@drpq kamezawa]# taskset 01 cat 512Mfile > /dev/null
> [root@drpq kamezawa]# exit
> exit
> [root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
> 539738112

> 0
> [root@drpq kamezawa]# bash
> #enter group B
> [root@drpq kamezawa]# echo $$ > /opt/mem_control/group_B/tasks
> [root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
> 539738112
> 557056
> [root@drpq kamezawa]# grep size-64 /proc/slabinfo
> size-64(DMA)   0   0   64 240   1 : tunables 120   60   8 : slabdata   0   0   0
> size-64   48263 59760   64 240   1 : tunables 120   60   8 : slabdata   249   249   12
> # migrate_test mmaps 512Mfile and call system call move_pages(). and sleep.
> [root@drpq kamezawa]# ./migrate_test 512Mfile 1 &
> [1] 4108
> #At the end of migration,

Where can I find migrate_test?

> [root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
> 539738112
> 537706496
>
> #Wow, charge is twice ;)
> [root@drpq kamezawa]# grep size-64 /proc/slabinfo
> size-64(DMA)   0   0   64 240   1 : tunables 120   60   8 : slabdata   0   0   0
> size-64   81180 92400   64 240   1 : tunables 120   60   8 : slabdata   385   385   12
>
> #Kill migrate_test, because 512Mfile is unmapped, charge in group_B is dropped.
> [root@drpq kamezawa]# kill %1
> [root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
> 536936448
> 1458176
> [1]+ Terminated   ./migrate_test 512Mfile 1
>
> #Try drop caches again
> [root@drpq kamezawa]# echo 3 > /proc/sys/vm/drop_caches
> [root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
> 536920064
> 1097728
> #no change because charge in group_A is leaked.....
>
> [root@drpq kamezawa]# grep size-64 /proc/slabinfo
> size-64(DMA)   0   0   64 240   1 : tunables 120   60   8 : slabdata   0   0   0
> size-64   48137 60720   64 240   1 : tunables 120   60   8 : slabdata   253   253   210
> [root@drpq kamezawa]#
>
> ==
>
> -Kame

>

--
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

---

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [1/1] fix pag
Posted by Balbir Singh on Tue, 02 Oct 2007 15:34:40 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 chaged back again if *mapped*.
>   - This means that cgroup before migraion can be different from one after
>   migraion

>From the test case mentioned earlier, this happens because the task has moved from one cgroup to another, right?

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

OK. This is an interesting situation, we uncharge page cache only in __remove_from_page_cache(). This is a combination of task migration followed by page migration, which the memory controller does not handle very well at the moment.

>   This is memory leak.

Yes, it is.

> ==
> This is bad.

Absolutely!

> And migration can migrate *not mapped* pages in future by migration-by-kernel
> driven by memory-unplug and defragment-by-migration at el.
>
> 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.
>
> Obviously, mem_cgroup_isolate_pages() and this page migration, which
> copies page_cgroup from old page to new page, has race.
>
> There seem to be 3 ways for avoiding this race.
> A. take mem_group->lock while mem_cgroup_page_migration().
> B. isolate pc from mem_cgroup's LRU when we isolate page from zone's LRU.
> C. ignore non-LRU page at mem_cgroup_isolate_pages().
>
> This patch uses method (C.) and modifes mem_cgroup_isolate_pages() igonres
> !PageLRU pages.
>

The page(s) is(are) !PageLRU only during page migration right?

> Tested and worked well in ia64/NUMA box.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
> ---
> include/linux/memcontrol.h | 22 +++++++++++++++
> mm/memcontrol.c | 62 ++---
> mm/migrate.c | 13 +++++++--
> 3 files changed, 90 insertions(+), 7 deletions(-)
>
> Index: linux-2.6.23-rc8-mm2/include/linux/memcontrol.h
> ===
> --- linux-2.6.23-rc8-mm2.orig/include/linux/memcontrol.h
> +++ linux-2.6.23-rc8-mm2/include/linux/memcontrol.h
> @@ -43,8 +43,14 @@ extern unsigned long mem_cgroup_isolate_
>   struct mem_cgroup *mem_cont,
>   int active);
> extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem, gfp_t gfp_mask);
> +
> extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
>   gfp_t gfp_mask);

> +/* For handling page migration in proper way */
> +extern void mem_cgroup_prepare_migration(struct page *page);
> +extern void mem_cgroup_end_migration(struct page *page);
> +extern void mem_cgroup_page_migration(struct page *newpage, struct page *page);
> +
>
> static inline struct mem_cgroup *mm_cgroup(const struct mm_struct *mm)
> {
> @@ -107,6 +113,22 @@ static inline struct mem_cgroup *mm_cgro
>   return NULL;
> }
>
> +/* For page migration */
> +static inline void mem_cgroup_prepare_migration(struct page *page)
> +{
> + return;
> +}
> +
> +static inline void mem_cgroup_end_migration(struct mem_cgroup *cgroup)
> +{
> + return;
> +}
> +
> +static inline void
> +mem_cgroup_page_migration(struct page *newpage, struct page *page)
> +{
> + return;
> +}
> #endif /* CONFIG_CGROUP_MEM_CONT */
>
> #endif /* _LINUX_MEMCONTROL_H */
> Index: linux-2.6.23-rc8-mm2/mm/memcontrol.c
> ===
> --- linux-2.6.23-rc8-mm2.orig/mm/memcontrol.c
> +++ linux-2.6.23-rc8-mm2/mm/memcontrol.c
> @@ -198,7 +198,7 @@ unsigned long mem_cgroup_isolate_pages(u
>   unsigned long scan;
>   LIST_HEAD(pc_list);
>   struct list_head *src;
> - struct page_cgroup *pc;
> + struct page_cgroup *pc, *tmp;
>
>   if (active)
>   src = &mem_cont->active_list;
> @@ -206,8 +206,10 @@ unsigned long mem_cgroup_isolate_pages(u
>   src = &mem_cont->inactive_list;
>
>   spin_lock(&mem_cont->lru_lock);

> - for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
> - pc = list_entry(src->prev, struct page_cgroup, lru);
> + scan = 0;
> + list_for_each_entry_safe_reverse(pc, tmp, src, lru) {
> + if (scan++ >= nr_to_scan)
> + break;
> page = pc->page;
> VM_BUG_ON(!pc);
>
> @@ -225,9 +227,14 @@ unsigned long mem_cgroup_isolate_pages(u
> /*
> * Reclaim, per zone
> * TODO: make the active/inactive lists per zone
> + * !PageLRU page can be found under us while migration.
> + * just ignore it.
> */
> - if (page_zone(page) != z)
> + if (page_zone(page) != z || !PageLRU(page)) {

I would prefer to do unlikely(!PageLRU(page)), since most of the
times the page is not under migration

> + /* Skip this */
> + /* Don't decrease scan here for avoiding dead lock */

Could we merge the two comments to one block comment?

> continue;
> + }
>
> /*
> * Check if the meta page went away from under us
> @@ -417,8 +424,14 @@ void mem_cgroup_uncharge(struct page_cgr
> return;
>
> if (atomic_dec_and_test(&pc->ref_cnt)) {
> +retry:
> page = pc->page;
> lock_page_cgroup(page);
> + /* migration occur ? */
> + if (page_get_page_cgroup(page) != pc) {
> + unlock_page_cgroup(page);
> + goto retry;

Shouldn't we check if page_get_page_cgroup(page) returns
NULL, if so, unlock and return?

> + }

```
>    mem = pc->mem_cgroup;
>    css_put(&mem->css);
>    page_assign_page_cgroup(page, NULL);
> @@ -432,6 +445,47 @@ void mem_cgroup_uncharge(struct page_cgr
>   }
>  }
>
> +void mem_cgroup_prepare_migration(struct page *page)
> +{
> + struct page_cgroup *pc;
> + lock_page_cgroup(page);
> + pc = page_get_page_cgroup(page);
> + if (pc)
> +  atomic_inc(&pc->ref_cnt);
> + unlock_page_cgroup(page);
> + return;
> +}
> +
> +void mem_cgroup_end_migration(struct page *page)
> +{
> + struct page_cgroup *pc = page_get_page_cgroup(page);
> + mem_cgroup_uncharge(pc);
> +}
> +
> +void mem_cgroup_page_migration(struct page *newpage, struct page *page)
> +{
> + struct page_cgroup *pc;
> + /*
> + * We already keep one reference to paage->cgroup.
> + * and both pages are guaranteed to be locked under page migration.
> + */
> + lock_page_cgroup(page);
> + lock_page_cgroup(newpage);
> + pc = page_get_page_cgroup(page);
> + if (!pc)
> +  goto unlock_out;
> + page_assign_page_cgroup(page, NULL);
> + pc->page = newpage;
> + page_assign_page_cgroup(newpage, pc);
> +
> +unlock_out:
> + unlock_page_cgroup(newpage);
> + unlock_page_cgroup(page);
> + return;
> +}
> +
> +
> +
```

> int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
> {
>   *tmp = memparse(buf, &buf);
> Index: linux-2.6.23-rc8-mm2/mm/migrate.c
> ===
> --- linux-2.6.23-rc8-mm2.orig/mm/migrate.c
> +++ linux-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(newpage, page);
>   remove_migration_ptes(page, newpage);
> - else
> + } else
>   newpage->mapping = NULL;
>
>   unlock_page(newpage);
> @@ -651,6 +652,8 @@ static int unmap_and_move(new_page_t get
>   rcu_read_lock();
>   rcu_locked = 1;
>   }
> + mem_cgroup_prepare_migration(page);
> +
>   /*
>   * This is a corner case handling.
>   * When a new swap-cache is read into, it is linked to LRU
> @@ -666,8 +669,12 @@ static int unmap_and_move(new_page_t get
>   if (!page_mapped(page))
>   rc = move_to_new_page(newpage, page);
>
> - if (rc)
> + if (rc) {
>   remove_migration_ptes(page, page);
> + mem_cgroup_end_migration(page);
> + } else
> + mem_cgroup_end_migration(newpage);
> +
> rcu_unlock:
>   if (rcu_locked)
>   rcu_read_unlock();
>

Looks good so far, even though I am yet to test. It looks like a tough problem to catch and debug. Thanks!

--
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

---

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [2/1] additio
Posted by Balbir Singh on Tue, 02 Oct 2007 15:36:27 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> The patch I sent needs following fix, sorry.
> Anyway, I'll repost good-version with reflected comments again.
>
> Thanks,
> -Kame

Just saw this now, I'll apply both the fixes, but it would be helpful
if you could post, one combined patch.

--
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

---

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [2/1] additio
Posted by KAMEZAWA Hiroyuki on Wed, 03 Oct 2007 00:29:09 GMT
View Forum Message <> Reply to Message

On Tue, 02 Oct 2007 21:06:27 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:

> > The patch I sent needs following fix, sorry.
> > Anyway, I'll repost good-version with reflected comments again.
> >
> > Thanks,
> > -Kame
>
> Just saw this now, I'll apply both the fixes, but it would be helpful
> if you could post, one combined patch.
>
Yes, I'll post refleshed easy-to-review version again.
I'm now planing to post a patch against next -mm.

Thanks,
-Kame

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [0/1]
Posted by KAMEZAWA Hiroyuki on Wed, 03 Oct 2007 00:39:32 GMT
View Forum Message <> Reply to Message

On Tue, 02 Oct 2007 19:04:15 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:
> > Current implementation of memory cgroup controller does following in migration.
> >
> > 1. uncharge when unmapped.
> > 2. charge again when remapped.
> >
> > Consider migrate a page from OLD to NEW.
> >
> > In following case, memory (for page_cgroup) will leak.
> >
> > 1. charge OLD page as page-cache. (charge = 1
> > 2. A process mmap OLD page. (chage + 1 = 2)
> > 3. A process migrates it.
> >   try_to_unmap(OLD) (charge - 1 = 1)
> >   replace OLD with NEW
> >   remove_migration_pte(NEW) (New is newly charged.)
> >   discard OLD page. (page_cgroup for OLD page is not reclaimed.)
> >
>

> Interesting test scenario, I'll try and reproduce the problem here.
> Why does discard OLD page not reclaim page_cgroup?
Just because OLD page is not page-cache at discarding. (it is replaced with NEW page)

>
> > [root@drpq kamezawa]# ./migrate_test 512Mfile 1 &
> > [1] 4108
> > #At the end of migration,
>
> Where can I find migrate_test?
>
here, (just I wrote for this test. works on my RHEL5/2.6.18-rc8-mm2/ia64 NUMA box)
This program doesn't check 'where is file cache ?' before migration. So please
check it by yourself before run.
==

```
#include <stdio.h>
#include <stdlib.h>
#include <syscall.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <numaif.h>
#include <unistd.h>

#define PAGESIZE (16384)

static inline int move_pages(pid_t pid, unsigned long nr_pages,
  const void **address,
  const int *nodes, int *status, int flags)
{
 return syscall(SYS_move_pages, pid, nr_pages, address,
   nodes, status, flags);
}
/*
 * migrate_test.c -- mmap file and migrate it to specified node.
 * %migrate_task file nodeid
 *
 */

int main(int argc, char *argv[])
{
 int ret, fd, val, node[1], status[1];
 char *addr, *c;
 unsigned long size, nr_pages, pos;
```

```
 struct stat statbuf;
 void *address[1];
 int target;

 if (argc != 3) {
  fprintf(stderr,"usage: migrate_test file node\n");
  exit(0);
 }
 target = atoi(argv[2]);

 fd = open(argv[1], O_RDONLY);
 if (fd < 0) {
  perror("open");
  exit(1);
 }

 ret = fstat(fd, &statbuf);
 if (ret < 0) {
  perror("fstat");
  exit(1);
 }

 size = statbuf.st_size;
 nr_pages = size/PAGESIZE;
 size = nr_pages * PAGESIZE;

 addr = mmap(NULL, size, PROT_READ, MAP_SHARED, fd, 0);

 if (addr == MAP_FAILED) {
  perror("mmap");
  exit(1);
 }
 /* Touch all */
 for (pos = 0; pos < nr_pages; pos++) {
  c = addr + pos * PAGESIZE;
  val += *c;
 }
 /* Move Pages */
 for (pos = 0; pos < nr_pages; pos++) {
  node[0] = target;
  status[0] = 0;
  address[0] = addr + pos * PAGESIZE;
  ret = move_pages(0, 1, address, node, status,
    MPOL_MF_MOVE_ALL);
  if (ret) {
   perror("move_pages");
  }
#if 1
```

```
  printf("pos %d %p %d %d\n",pos, address[0], node[0], status[0]);
#endif
 }
 /* Touch all again....maybe unnecessary.*/
 for (pos = 0; pos < nr_pages; pos++) {
  c = addr + pos * PAGESIZE;
  val += *c;
 }
 while (1) {
  /* mmap until killed */
  pause();
 }
 printf("val %d\n",val);
 return 0;
}
```

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [1/1] fix pag
Posted by KAMEZAWA Hiroyuki on Wed, 03 Oct 2007 00:53:09 GMT
View Forum Message <> Reply to Message

On Tue, 02 Oct 2007 21:04:40 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> 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 chaged back again if *mapped*.
> >   - This means that cgroup before migraion can be different from one after
> >   migraion
>
> >From the test case mentioned earlier, this happens because the task has
> moved from one cgroup to another, right?
Ah, yes.

> > And migration can migrate *not mapped* pages in future by migration-by-kernel
> > driven by memory-unplug and defragment-by-migration at el.
> >
> > 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.
> >
> > Obviously, mem_cgroup_isolate_pages() and this page migration, which
> > copies page_cgroup from old page to new page, has race.
> >
> > There seem to be 3 ways for avoiding this race.
> > A. take mem_group->lock while mem_cgroup_page_migration().
> > B. isolate pc from mem_cgroup's LRU when we isolate page from zone's LRU.
> > C. ignore non-LRU page at mem_cgroup_isolate_pages().
> >
> > This patch uses method (C.) and modifes mem_cgroup_isolate_pages() igonres
> > !PageLRU pages.
> >
>
> The page(s) is(are) !PageLRU only during page migration right?
>
Hmm...!PageLRU() means that page is not on LRU.
Then, kswapd can remove a page from LRU.

> > - if (page_zone(page) != z)
> > + if (page_zone(page) != z || !PageLRU(page)) {
>
> I would prefer to do unlikely(!PageLRU(page)), since most of the
> times the page is not under migration
>
I see.

> > +   /* Skip this */
> > +   /* Don't decrease scan here for avoiding dead lock */
>
> Could we merge the two comments to one block comment?
>
will do

> >   continue;
> > + }
> >
> >   /*
> >   * Check if the meta page went away from under us

> > @@ -417,8 +424,14 @@ void mem_cgroup_uncharge(struct page_cgr
> > return;
> >
> > if (atomic_dec_and_test(&pc->ref_cnt)) {
> > +retry:
> > page = pc->page;
> > lock_page_cgroup(page);
> > + /* migration occur ? */
> > + if (page_get_page_cgroup(page) != pc) {
> > + unlock_page_cgroup(page);
> > + goto retry;
>
> Shouldn't we check if page_get_page_cgroup(page) returns
> NULL, if so, unlock and return?
Hmm, I think page_get_page_cgroup(page) != pc covers it. pc is not NULL.

Thanks,
-Kame

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

---

Subject: Re: [BUGFIX][RFC][PATCH][only -mm] FIX memory leak in memory cgroup vs. page migration [0/1]
Posted by Balbir Singh on Tue, 16 Oct 2007 21:03:28 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
[snip]
> # migrate_test mmaps 512Mfile and call system call move_pages(). and sleep.
> [root@drpq kamezawa]# ./migrate_test 512Mfile 1 &
> [1] 4108

This step fails for me. I get an -ENOENT error from the utility you sent me. As I look through the migration code more (It's too late for me to double check), but it seems that only pages mapped in the process are migrated. cat(1) won't really map anything. Am I missing some of the reproduction steps?

> #At the end of migration,
> [root@drpq kamezawa]# cat /opt/mem_control/group_?/memory.usage_in_bytes
> 539738112
> 537706496
>
> #Wow, charge is twice ;)

--
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