Subject: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [0/5]
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 04:53:45 GMT
View Forum Message <> Reply to Message

This set is a fix for memory cgroup against 2.6.23-rc8-mm2.
Not including any new feature.

If this is merged to the next -mm, I'm happy.

Patches:
[1/5] ... fix refcnt handling in charge mem_cgroup_charge()
[2/5] ... fix error handling path in mem_cgroup_charge()
[3/5] ... check page->cgroup under lock again.
[4/5] ... fix mem_cgroup_isolate_pages() to skip !PageLRU() pages.
[5/5] ... fix page migration under memory controller, fixes leak.

Changes from previous ones.
 -- dropped new feature.... force_empty patch. It will be posted later.
 -- fix typos
 -- added comments

Tested on x86-64/fake-NUMA system.

Thanks,
-Kame

Subject: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [1/5] -- fix refcnt check in mem_cgroup_charge
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 04:55:49 GMT
View Forum Message <> Reply to Message

The logic of uncharge is
 - decrement refcnt -> lock page cgroup -> remove page cgroup.
But the logic of charging is
 - lock page cgroup -> increment refcnt -> return.

Then, one charge will be added to a page_cgroup under being removed.
This makes no big trouble (like panic) but one charge is lost.

This patch add a test at charging to verify page_cgroup's refcnt is
greater than 0. If not, unlock and retry.

Changelog v1->v2:
 * added cpu_relax() before retry.

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

 mm/memcontrol.c | 10 ++++++++--
 1 file changed, 8 insertions(+), 2 deletions(-)

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
@@ -271,14 +271,20 @@ int mem_cgroup_charge(struct page *page,
  * to see if the cgroup page already has a page_cgroup associated
  * with it
  */
+retry:
 lock_page_cgroup(page);
 pc = page_get_page_cgroup(page);
 /*
  * The page_cgroup exists and the page has already been accounted
  */
 if (pc) {
-	atomic_inc(&pc->ref_cnt);
-	goto done;
+	if (unlikely(!atomic_inc_not_zero(&pc->ref_cnt))) {
+		/* this page is under being uncharged ? */
+		unlock_page_cgroup(page);
+		cpu_relax();
+		goto retry;
+	} else
+		goto done;
 }

 unlock_page_cgroup(page);

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

Subject: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [2/5] -- fix error handling path of mem_cgroup_c
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 04:57:09 GMT
View Forum Message <> Reply to Message

This unlock_page_cgroup() is unnecessary.

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

mm/memcontrol.c | 2 --
1 file changed, 2 deletions(-)

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
@@ -382,9 +382,7 @@ done:
return 0;
free_pc:
kfree(pc);
- return -ENOMEM;
err:
- unlock_page_cgroup(page);
return -ENOMEM;
}

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

Subject: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [3/5] --- add helper function for assingin page_
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 05:00:10 GMT
View Forum Message <> Reply to Message

This patch adds following functions.
- clear_page_cgroup(page, pc)
- page_cgroup_assign_new_page_group(page, pc)

Mainly for cleanup.

A manner "check page->cgroup again after lock_page_cgroup()" is
implemented in straight way.

Changelog v1 -> v2:
  - added comments.
  - adder printk in mem_cgroup_uncharge() error path, but this is planned to be
    removed by other patch
Note:
  - a comment in mem_cgroup_uncharge() will be removed by force-empty patch

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

 mm/memcontrol.c | 100 ++----------
 1 file changed, 73 insertions(+), 27 deletions(-)

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
@@ -162,6 +162,48 @@ static void __always_inline unlock_page_
  bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
 }

+/*
+ * Tie new page_cgroup to struct page under lock_page_cgroup()
+ * This can fail if the page has been tied to a page_cgroup.
+ * If success, returns 0.
+ */
+static inline int
+page_cgroup_assign_new_page_cgroup(struct page *page, struct page_cgroup *pc)
+{
+ int ret = 0;
+
+ lock_page_cgroup(page);
+ if (!page_get_page_cgroup(page))
+ page_assign_page_cgroup(page, pc);
+ else /* A page is tied to other pc. */
+ ret = 1;
+ unlock_page_cgroup(page);
+ return ret;
+}
+
+/*
+ * Clear page->page_cgroup member under lock_page_cgroup().
+ * If given "pc" value is different from one page->page_cgroup,
+ * page->cgroup is not cleared.
+ * Returns a value of page->page_cgroup at lock taken.
+ * A can can detect failure of clearing by following
+ * clear_page_cgroup(page, pc) == pc

```
+ */
+
+static inline struct page_cgroup *
+clear_page_cgroup(struct page *page, struct page_cgroup *pc)
+{
+ struct page_cgroup *ret;
+ /* lock and clear */
+ lock_page_cgroup(page);
+ ret = page_get_page_cgroup(page);
+ if (likely(ret == pc))
+  page_assign_page_cgroup(page, NULL);
+ unlock_page_cgroup(page);
+ return ret;
+}
+
+
 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
  if (active)
@@ -260,7 +302,7 @@ int mem_cgroup_charge(struct page *page,
     gfp_t gfp_mask)
 {
  struct mem_cgroup *mem;
- struct page_cgroup *pc, *race_pc;
+ struct page_cgroup *pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;

@@ -354,24 +396,20 @@ noreclaim:
   goto free_pc;
  }

- lock_page_cgroup(page);
- /*
-  * Check if somebody else beat us to allocating the page_cgroup
-  */
- race_pc = page_get_page_cgroup(page);
- if (race_pc) {
-  kfree(pc);
-  pc = race_pc;
-  atomic_inc(&pc->ref_cnt);
-  res_counter_uncharge(&mem->res, PAGE_SIZE);
-  css_put(&mem->css);
-  goto done;
- }
-
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
```

```
  pc->page = page;
- page_assign_page_cgroup(page, pc);
+ if (page_cgroup_assign_new_page_cgroup(page, pc)) {
+  /*
+   * an another charge is added to this page already.
+   * we do take lock_page_cgroup(page) again and read
+   * page->cgroup, increment refcnt.... just retry is OK.
+   */
+  res_counter_uncharge(&mem->res, PAGE_SIZE);
+  css_put(&mem->css);
+  kfree(pc);
+  goto retry;
+ }

  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
@@ -422,17 +460,25 @@ 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);
-
-   spin_lock_irqsave(&mem->lru_lock, flags);
-   list_del_init(&pc->lru);
-   spin_unlock_irqrestore(&mem->lru_lock, flags);
-  kfree(pc);
+  /*
+   * get page->cgroup and clear it under lock.
+   */
+  if (clear_page_cgroup(page, pc) == pc) {
+   mem = pc->mem_cgroup;
+   css_put(&mem->css);
+   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 {
+   /*
+    * Note:This will be removed when force-empty patch is
+    * applied. just show warning here.
+    */
+   printk(KERN_ERR "Race in mem_cgroup_uncharge() ?");
```

```
+   dump_stack();
+ }
  }
 }
```

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

---

Subject: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [4/5] skip !PageLRU page in mem_cgroup_isolate_p
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 05:01:15 GMT
View Forum Message <> Reply to Message

This patch makes mem_cgroup_isolate_pages() to be

  - ignore !PageLRU pages.
  - fixes the bug that isolation makes no progress if page_zone(page) != zone
    page once find. (just increment scan in this case.)

kswapd and memory migration removes a page from list when it handles
a page for reclaiming/migration.

Because __isolate_lru_page() doesn't moves page !PageLRU pages, it will
be safe to avoid touching !PageLRU() page and its page_cgroup.

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

 mm/memcontrol.c | 13 ++++++++++---
 1 file changed, 10 insertions(+), 3 deletions(-)

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
@@ -240,7 +240,7 @@ 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;
@@ -248,11 +248,18 @@ 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_taken)
+   break;
   page = pc->page;
   VM_BUG_ON(!pc);

+  if (unlikely(!PageLRU(page))) {
+   scan--;
+   continue;
+  }
+
   if (PageActive(page) && !active) {
    __mem_cgroup_move_lists(pc, true);
    scan--;
```

Subject: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [5/5] --- fix page migration under memory contro
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 05:02:20 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 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 is not on LRU.

This 3 facts guarantees page_cgroup() migration has no race.

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_cgro
  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
@@ -488,6 +488,49 @@ 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)
 {
```

Subject: Re: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [5/5] --- fix page migration under memory co

Posted by Balbir Singh on Thu, 11 Oct 2007 12:05:40 GMT

View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:

```
> +
> +static inline void
> +mem_cgroup_page_migration(struct page *page, struct page *newpage);
```

Typo, the semicolon needs to go :-)

```
> +{
> +}
> +
> +
```

--
 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: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [5/5] --- fix page migration under memory co
Posted by KAMEZAWA Hiroyuki on Thu, 11 Oct 2007 12:54:13 GMT
View Forum Message <> Reply to Message

On Thu, 11 Oct 2007 17:35:40 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:
> > +
> > +static inline void
> > +mem_cgroup_page_migration(struct page *page, struct page *newpage);
>
> Typo, the semicolon needs to go :-)
>
Oh, thanks!, will send updated version later.

-Kame

_____

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

---

Subject: Re: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [0/5]
Posted by Balbir Singh on Thu, 11 Oct 2007 19:56:46 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This set is a fix for memory cgroup against 2.6.23-rc8-mm2.
> Not including any new feature.
>
> If this is merged to the next -mm, I'm happy.
>
> Patches:
> [1/5] ... fix refcnt handling in charge mem_cgroup_charge()
> [2/5] ... fix error handling path in mem_cgroup_charge()
> [3/5] ... check page->cgroup under lock again.

> [4/5] ... fix mem_cgroup_isolate_pages() to skip !PageLRU() pages.
> [5/5] ... fix page migration under memory controller, fixes leak.
>
> Changes from previous ones.
> -- dropped new feature.... force_empty patch. It will be posted later.
> -- fix typos
> -- added comments
>
> Tested on x86-64/fake-NUMA system.
>

I tested the patches, ran kernbench, lmbench and some tests with
parallel containers. Except for the one typo in the page migration
patch, the patches worked quite well. KAMEZAWA-San, could you please
send the updated patch with the compilation fix.

I am yet to test the migration fix (I am yet to get access to a NUMA/
box capable of fake NUMA). I have not measured the performance impact
of these patches.

Andrew, could you please consider these patches for -mm inclusion once
KAMEZAWA-San sends out the fixed migration patch.

> Thanks,
> -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: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [5/5] --- fix page migration under memory co
Posted by KAMEZAWA Hiroyuki on Fri, 12 Oct 2007 01:12:59 GMT
View Forum Message <> Reply to Message

Hi, here is fixed one.
thanks,
-Kame
==
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 is not on LRU.

This 3 facts guarantees page_cgroup() migration has no race.

Tested and worked well in x86_64/fake-NUMA box.
Changelog v2 -> v3
  - fixed typo in !CONFIG_CGROUP_MEM_CONT case.

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_cgro
  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
@@ -488,6 +488,49 @@ 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)
 {
```

Subject: Re: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [4/5] skip !PageLRU page in mem_cgroup_isola
Posted by KAMEZAWA Hiroyuki on Tue, 30 Oct 2007 05:47:45 GMT
View Forum Message <> Reply to Message

I'm sorry that this patch needs following fix..
Andrew, could you apply this ?
(All version I sent has this bug....Sigh)

Thanks,
-Kame
==
Bugfix for memory cgroup skip !PageLRU page in mem_cgroup_isolate_pages

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

Index: devel-2.6.23-mm1/mm/memcontrol.c
===
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -260,7 +260,7 @@ unsigned long mem_cgroup_isolate_pages(u
  spin_lock(&mem_cont->lru_lock);
  scan = 0;
  list_for_each_entry_safe_reverse(pc, tmp, src, lru) {
- if (scan++ > nr_taken)
+ if (scan++ > nr_to_scan)
    break;
   page = pc->page;
   VM_BUG_ON(!pc);

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

Subject: Re: [PATCH][BUGFIX][for -mm] Misc fix for memory cgroup [4/5] skip !PageLRU page in mem_cgroup_isola
Posted by Balbir Singh on Tue, 30 Oct 2007 06:00:43 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> I'm sorry that this patch needs following fix..
> Andrew, could you apply this ?

> (All version I sent has this bug....Sigh)
>
> Thanks,
> -Kame
> ==
> Bugfix for memory cgroup skip !PageLRU page in mem_cgroup_isolate_pages
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
> Index: devel-2.6.23-mm1/mm/memcontrol.c
> ===
> --- devel-2.6.23-mm1.orig/mm/memcontrol.c
> +++ devel-2.6.23-mm1/mm/memcontrol.c
> @@ -260,7 +260,7 @@ unsigned long mem_cgroup_isolate_pages(u
>   spin_lock(&mem_cont->lru_lock);
>   scan = 0;
>   list_for_each_entry_safe_reverse(pc, tmp, src, lru) {
> - if (scan++ > nr_taken)
> + if (scan++ > nr_to_scan)
>   break;
>   page = pc->page;
>   VM_BUG_ON(!pc);
>

Good catch! Sorry, I missed it in the review

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