Subject: [PATCH][for -mm] Fix and Enhancements for memory cgroup [0/6] intro
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 09:46:20 GMT
View Forum Message <> Reply to Message

Hi, Balbir-san
This is a patch set against memory cgroup I have now.
Reflected comments I got.

=
[1] charge refcnt fix patch - avoid charging against a page which is being uncharged.
[2] fix-err-handling patch - remove unnecesary unlock_page_cgroup()
[3] lock and page->cgroup patch - add helper function for charge/uncharge
[4] avoid handling no LRU patch - makes mem_cgroup_isolate_pages() avoid handling !Page_LRU pages.
[5] migration fix patch - a fix for page migration.
[6] force reclaim patch - add an interface for uncharging all pages in empty cgroup.
=

BTW, which way would you like to go ?

1. You'll merge this set (and my future patch) to your set as Memory Cgroup Maintainer and pass to Andrew Morton, later. And we'll work against your tree.
2. I post this set to the (next) -mm. And we'll work agaisnt -mm.

not as my usual patch, tested on x86-64 fake-NUMA.

Thanks,
-Kame

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

Subject: [PATCH][for -mm] Fix and Enhancements for memory cgroup [1/6] fix refcnt race in charge/uncharge
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 09:49:25 GMT
View Forum Message <> Reply to Message

The logic of uncharging 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.

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

```
 mm/memcontrol.c |    9 +++++++--
 1 file changed, 7 insertions(+), 2 deletions(-)

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
@@ -271,14 +271,19 @@ 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 uncharge ? */
+   unlock_page_cgroup(page);
+   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][for -mm] Fix and Enhancements for memory cgroup [2/6] fix err

handling in charging
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 09:50:18 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: 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
@@ -381,9 +381,7 @@ 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][for -mm] Fix and Enhancements for memory cgroup [3/6] add helper function for page_cgroup
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 09:51:32 GMT
View Forum Message <> Reply to Message

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

Mainly for cleaunp.

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

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

```
 mm/memcontrol.c |   76 ++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 49 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 +162,35 @@ static void __always_inline unlock_page_
  bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
 }

+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
+  ret = 1;
+ unlock_page_cgroup(page);
+ return ret;
+}
+
+
+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 +289,7 @@ 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;

@@ -353,24 +382,16 @@ 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)) {
+  /* race ... undo and retry */
+  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);
@@ -421,17 +442,18 @@ 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);
+  /*
+   * Obetaion 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);
+  }
  }
 }
```

_____

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

---

Subject: [PATCH][for -mm] Fix and Enhancements for memory cgroup [4/6] avoid handling !LRU page in mem_cgrou
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 09:53:10 GMT
View Forum Message <> Reply to Message

This patch makes mem_cgroup_isolate_pages() to be

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

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

__isolate_lru_page() doesn't moves page !PageLRU pages, then, it will
be safe to avoid touching the 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
@@ -227,7 +227,7 @@ 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;
@@ -235,11 +235,18 @@ 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--;
```

_____

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

---

Subject: [PATCH][for -mm] Fix and Enhancements for memory cgroup [5/6] memory cgroup and migration fix
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 09:54:31 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 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 patche 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
@@ -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 rece with uncharge() routines becasue 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)
 {
```

_____

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

---

Subject: [PATCH][for -mm] Fix and Enhancements for memory cgroup [6/6] add force reclaim interface

This patch adds an interface "memory.force_reclaim".
Any write to this file will drop all charges in this cgroup if
there is no task under.

%echo 1 > /...../memory.force_reclaim

will drop all charges of memory cgroup if cgroup's tasks is empty.

This is useful to invoke rmdir() against memory cgroup successfully.

Tested and worked well on x86_64/fake-NUMA system.

Changelog:
  - added a new interface force_relcaim.
  - changes spin_lock to spin_lock_irqsave().

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

 mm/memcontrol.c | 79
++
 1 file changed, 79 insertions(+)

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
@@ -507,6 +507,55 @@ retry:
  return;
 }

+static void
+mem_cgroup_force_reclaim_list(struct mem_cgroup *mem, struct list_head *list)
+{
+ struct page_cgroup *pc;
+ struct page *page;
+ int count = SWAP_CLUSTER_MAX;
+ unsigned long flags;
+
+ spin_lock_irqsave(&mem->lru_lock, flags);
+
+ while (!list_empty(list)) {
+ pc = list_entry(list->prev, struct page_cgroup, lru);
+ page = pc->page;
+ if (clear_page_cgroup(page, pc) == pc) {

```
+   css_put(&mem->css);
+   res_counter_uncharge(&mem->res, PAGE_SIZE);
+   list_del_init(&pc->lru);
+   kfree(pc);
+  } else
+   count = 1; /* race? ...do relax */
+
+  if (--count == 0) {
+   spin_unlock_irqrestore(&mem->lru_lock, flags);
+   cond_resched();
+   spin_lock_irqsave(&mem->lru_lock, flags);
+   count = SWAP_CLUSTER_MAX;
+  }
+ }
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+}
+
+int mem_cgroup_force_reclaim(struct mem_cgroup *mem)
+{
+ int ret = -EBUSY;
+ while (!list_empty(&mem->active_list) ||
+        !list_empty(&mem->inactive_list)) {
+  if (atomic_read(&mem->css.cgroup->count) > 0)
+   goto out;
+  mem_cgroup_force_reclaim_list(mem, &mem->active_list);
+  mem_cgroup_force_reclaim_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)
 {
  *tmp = memparse(buf, &buf);
@@ -592,6 +641,31 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
 }

+
+static ssize_t mem_force_reclaim_write(struct cgroup *cont,
+    struct cftype *cft, struct file *file,
+    const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
```

```
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ int ret;
+ ret = mem_cgroup_force_reclaim(mem);
+ if (!ret)
+  ret = nbytes;
+ return ret;
+}
+
+static ssize_t mem_force_reclaim_read(struct cgroup *cont,
+    struct cftype *cft,
+    struct file *file, char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ char buf[2] = "0";
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+   ppos, buf, strlen(buf));
+}
+
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -614,6 +688,11 @@ static struct cftype mem_cgroup_files[]
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
+ {
+  .name = "force_reclaim",
+  .write = mem_force_reclaim_write,
+  .read = mem_force_reclaim_read,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
```


Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [0/6] intro
Posted by Balbir Singh on Tue, 09 Oct 2007 10:30:55 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:

> Hi, Balbir-san
> This is a patch set against memory cgroup I have now.
> Reflected comments I got.
>
> =
> [1] charge refcnt fix patch - avoid charging against a page which is being
> uncharged.
> [2] fix-err-handling patch - remove unnecesary unlock_page_cgroup()
> [3] lock and page->cgroup patch - add helper function for charge/uncharge
> [4] avoid handling no LRU patch - makes mem_cgroup_isolate_pages() avoid
> handling !Page_LRU pages.
> [5] migration fix patch - a fix for page migration.
> [6] force reclaim patch - add an interface for uncharging all pages in
> empty cgroup.
> =
>

Thank you very much for working on this.

> BTW, which way would you like to go ?
>
> 1. You'll merge this set (and my future patch) to your set as
> Memory Cgroup Maintainer and pass to Andrew Morton, later.
> And we'll work against your tree.
> 2. I post this set to the (next) -mm. And we'll work agaisnt -mm.
>

I think (2) is better. I don't maintain my own tree, so lets get
all the fixes and enhancements into -mm

> not as my usual patch, tested on x86-64 fake-NUMA.
>

I'll also test these patches.

> Thanks,
> -Kame
>

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [1/6] fix refcnt race in charge/uncharge
Posted by Balbir Singh on Tue, 09 Oct 2007 10:38:11 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> The logic of uncharging 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.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 9 +++++++--
> 1 file changed, 7 insertions(+), 2 deletions(-)
>
> 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
> @@ -271,14 +271,19 @@ 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 uncharge ? */
> +   unlock_page_cgroup(page);
> +   goto retry;
> + } else

> +   goto done;
>   }
>
>   unlock_page_cgroup(page);
>
>

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

---

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [2/6] fix err handling in charging
Posted by Balbir Singh on Tue, 09 Oct 2007 10:48:08 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> 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: 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
> @@ -381,9 +381,7 @@ done:
>   return 0;
> free_pc:
>   kfree(pc);
> - return -ENOMEM;
> err:
> - unlock_page_cgroup(page);
>   return -ENOMEM;

> }
>
>

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

---

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [3/6] add helper function for page_cgrou
Posted by Balbir Singh on Tue, 09 Oct 2007 11:09:48 GMT

View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch adds follwoing functions.
>   - clear_page_cgroup(page, pc)
>   - page_cgroup_assign_new_page_group(page, pc)
>
> Mainly for cleaunp.
>
> A manner "check page->cgroup again after lock_page_cgroup()" is
> implemented in straight way.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
>
> mm/memcontrol.c | 76 ++--------------------------------------
> 1 file changed, 49 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 +162,35 @@ static void __always_inline unlock_page_
>   bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
> }

>
> +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
> + ret = 1;
> + unlock_page_cgroup(page);
> + return ret;
> +}
> +

Some comment on when the assignment can fail, for example if page
already has a page_cgroup associated with it, would be nice.

> +
> +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;
> +}
> +

We could add a comment stating that clearing would fail if the page's
cgroup is not pc

> +
> static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
> {
>   if (active)
> @@ -260,7 +289,7 @@ 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;
>
> @@ -353,24 +382,16 @@ 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)) {
> + /* race ... undo and retry */
> + res_counter_uncharge(&mem->res, PAGE_SIZE);
> + css_put(&mem->css);
> + kfree(pc);
> + goto retry;

This part is a bit confusing, why do we want to retry. If someone
else charged the page already, we just continue, we let the other
task take the charge and add this page to it's cgroup

> + }
>
>   spin_lock_irqsave(&mem->lru_lock, flags);
>   list_add(&pc->lru, &mem->active_list);
> @@ -421,17 +442,18 @@ 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);
> + /*
> + * Obetaion page->cgroup and clear it under lock.
^^^^^^^^
Not sure if I've come across this word before

> + */
> + if (clear_page_cgroup(page, pc) == pc) {

OK.. so we've come so far and seen that pc has changed underneath us,
what do we do with this 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);
> + }
> }
> }
>
>

--
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][for -mm] Fix and Enhancements for memory cgroup [3/6] add helper function for page_cgrou
Posted by KAMEZAWA Hiroyuki on Tue, 09 Oct 2007 11:26:42 GMT
View Forum Message <> Reply to Message

On Tue, 09 Oct 2007 16:39:48 +0530

Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > +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
> > + ret = 1;
> > + unlock_page_cgroup(page);
> > + return ret;
> > +}
> > +
>
> Some comment on when the assignment can fail, for example if page
> already has a page_cgroup associated with it, would be nice.
>
Sure ,will add.

> > +
> > +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;
> > +}
> > +
>
> We could add a comment stating that clearing would fail if the page's
> cgroup is not pc
>
will add, too.

> > +
> > static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
> > {
> >   if (active)
> > @@ -260,7 +289,7 @@ 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;
> >
> > @@ -353,24 +382,16 @@ 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)) {
> > + /* race ... undo and retry */
> > + res_counter_uncharge(&mem->res, PAGE_SIZE);
> > + css_put(&mem->css);
> > + kfree(pc);
> > + goto retry;
>
> This part is a bit confusing, why do we want to retry. If someone
> else charged the page already, we just continue, we let the other
> task take the charge and add this page to it's cgroup
>
Okay. will add precise text.

> > + }
> >
> >   spin_lock_irqsave(&mem->lru_lock, flags);
> >   list_add(&pc->lru, &mem->active_list);
> > @@ -421,17 +442,18 @@ 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);
> > + /*
> > + * Obetaion page->cgroup and clear it under lock.
> ^^^^^^^^
> Not sure if I've come across this word before
Sorry (>_<;
Get page->cgroup and clear it under lock.

>
> > + */
> > + if (clear_page_cgroup(page, pc) == pc) {
>
> OK.. so we've come so far and seen that pc has changed underneath us,
> what do we do with this pc?
>
Hmm... How about this ?
==
 if (clear_page_cgroup(page, pc) == pc) {
 /* do usual work */
 } else {
 BUG();
 }
== or BUG_ON(clear_page_cgroup(page, pc) != pc)

I have no clear idea when this race will occur.
But this "lock and clear" behavior is sane, I think.

Thanks,
-kame

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

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [4/6] avoid handling !LRU page in mem_c
Posted by Balbir Singh on Tue, 09 Oct 2007 15:35:15 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch makes mem_cgroup_isolate_pages() to be
>
>   - ignore !PageLRU pages.
>   - fixes the bug that it makes no progress if page_zone(page) != zone
>   page once find. (just increment scan in this case.)
>
> kswapd and memory migraion removes a page from list when it handles
> a page for reclaiming/migration.
>
> __isolate_lru_page() doesn't moves page !PageLRU pages, then, it will
> be safe to avoid touching the 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
> @@ -227,7 +227,7 @@ 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;
> @@ -235,11 +235,18 @@ 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--;
>

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

---

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 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 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 patche 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
> @@ -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 rece with uncharge() routines becasue 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

---

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [6/6] add force reclaim interface

Posted by Balbir Singh on Tue, 09 Oct 2007 18:44:53 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch adds an interface "memory.force_reclaim".
> Any write to this file will drop all charges in this cgroup if
> there is no task under.

>
> %echo 1 > /...../memory.force_reclaim
>

Looks like a good name, do you think system administrators would
find force_empty more useful?

> will drop all charges of memory cgroup if cgroup's tasks is empty.
>
> This is useful to invoke rmdir() against memory cgroup successfully.
>
> Tested and worked well on x86_64/fake-NUMA system.
>
> Changelog:
>   - added a new interface force_relcaim.
>   - changes spin_lock to spin_lock_irqsave().
>
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 79
++
> 1 file changed, 79 insertions(+)
>
> 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
> @@ -507,6 +507,55 @@ retry:
>   return;
> }
>
> +static void
> +mem_cgroup_force_reclaim_list(struct mem_cgroup *mem, struct list_head *list)
> +{
> + struct page_cgroup *pc;
> + struct page *page;
> + int count = SWAP_CLUSTER_MAX;
> + unsigned long flags;
> +
> + spin_lock_irqsave(&mem->lru_lock, flags);
> +

Can we add a comment here stating that this routine reclaims just
from the per cgroup LRU and not from the zone LRU to which the
page belongs.

```
> + while (!list_empty(list)) {
> +  pc = list_entry(list->prev, struct page_cgroup, lru);
> +  page = pc->page;
> +  if (clear_page_cgroup(page, pc) == pc) {
> +   css_put(&mem->css);
> +   res_counter_uncharge(&mem->res, PAGE_SIZE);
> +   list_del_init(&pc->lru);
> +   kfree(pc);
> +  } else
> +   count = 1; /* race? ...do relax */
> +
> +  if (--count == 0) {
> +   spin_unlock_irqrestore(&mem->lru_lock, flags);
> +   cond_resched();
> +   spin_lock_irqsave(&mem->lru_lock, flags);
> +   count = SWAP_CLUSTER_MAX;
> +  }
> + }
> + spin_unlock_irqrestore(&mem->lru_lock, flags);
> +}
> +
> +int mem_cgroup_force_reclaim(struct mem_cgroup *mem)
> +{
> + int ret = -EBUSY;
> + while (!list_empty(&mem->active_list) ||
> +        !list_empty(&mem->inactive_list)) {
> +  if (atomic_read(&mem->css.cgroup->count) > 0)
> +   goto out;
> +  mem_cgroup_force_reclaim_list(mem, &mem->active_list);
> +  mem_cgroup_force_reclaim_list(mem, &mem->inactive_list);
> + }
> + ret = 0;
> +out:
> + css_put(&mem->css);
```

We do a css_put() here, did we do a css_get() anywhere?

```
> + return ret;
> +}
> +
> +
> +
>  int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
>  {
>   *tmp = memparse(buf, &buf);
> @@ -592,6 +641,31 @@ static ssize_t mem_control_type_read(str
>      ppos, buf, s - buf);
>  }
```

>
> +
> +static ssize_t mem_force_reclaim_write(struct cgroup *cont,
> +   struct cftype *cft, struct file *file,
> +   const char __user *userbuf,
> +   size_t nbytes, loff_t *ppos)
> +{
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> + int ret;
> + ret = mem_cgroup_force_reclaim(mem);
> + if (!ret)
> + ret = nbytes;
> + return ret;
> +}
> +
> +static ssize_t mem_force_reclaim_read(struct cgroup *cont,
> +   struct cftype *cft,
> +   struct file *file, char __user *userbuf,
> +   size_t nbytes, loff_t *ppos)
> +{
> + char buf[2] = "0";
> + return simple_read_from_buffer((void __user *)userbuf, nbytes,
> +   ppos, buf, strlen(buf));
> +}
> +
> +
> static struct cftype mem_cgroup_files[] = {
>   {
>   .name = "usage_in_bytes",
> @@ -614,6 +688,11 @@ static struct cftype mem_cgroup_files[]
>   .write = mem_control_type_write,
>   .read = mem_control_type_read,
>   },
> + {
> + .name = "force_reclaim",
> + .write = mem_force_reclaim_write,
> + .read = mem_force_reclaim_read,
> + },
> };
>
> static struct mem_cgroup init_mem_cgroup;
>
>

--
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][for -mm] Fix and Enhancements for memory cgroup [1/6] fix refcnt race in charge/uncharge
Posted by yamamoto on Tue, 09 Oct 2007 22:31:38 GMT
View Forum Message <> Reply to Message

> The logic of uncharging 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.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 9 +++++++--
> 1 file changed, 7 insertions(+), 2 deletions(-)
>
> 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
> @@ -271,14 +271,19 @@ 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 uncharge ? */
> + unlock_page_cgroup(page);

cpu_relax() here?

YAMAMOTO Takashi

> + 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: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [1/6] fix refcnt race in charge/uncharge
Posted by KAMEZAWA Hiroyuki on Wed, 10 Oct 2007 00:34:42 GMT
View Forum Message <> Reply to Message

On Wed, 10 Oct 2007 07:31:38 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> > - atomic_inc(&pc->ref_cnt);
> > - goto done;
> > + if (unlikely(!atomic_inc_not_zero(&pc->ref_cnt))) {
> > + /* this page is under being uncharge ? */
> > + unlock_page_cgroup(page);
>
> cpu_relax() here?
>
Ah, yes. there should be. I'll add.

Thanks,
-Kame

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

---

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [3/6] add helper function for page_cgrou

Posted by KAMEZAWA Hiroyuki on Wed, 10 Oct 2007 00:38:39 GMT
View Forum Message <> Reply to Message

On Tue, 9 Oct 2007 20:26:42 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> >
> > > + */
> > > + if (clear_page_cgroup(page, pc) == pc) {
> >
> > OK.. so we've come so far and seen that pc has changed underneath us,
> > what do we do with this pc?
> >
> Hmm... How about this ?
> ==
> if (clear_page_cgroup(page, pc) == pc) {
> /* do usual work */
> } else {
> BUG();
> }
> == or BUG_ON(clear_page_cgroup(page, pc) != pc)
>
> I have no clear idea when this race will occur.
After good sleep, I noticed there is a race with force_reclaim (in patch 6).

force_reclaim doesn't check refcnt before clearing page->pc.

My final view will be
==
  if (clear_page_cgroup(page, pc) == pc) {
 /* do usual work */
  } else {
 /* force reclaim clears page->page_cgroup */
  }
==
Anyway, I'll add a meaningful comment here.

Thanks,
-Kame

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

Subject: Re: [PATCH][for -mm] Fix and Enhancements for memory cgroup [6/6] add force reclaim interface

On Wed, 10 Oct 2007 00:14:53 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:
> > This patch adds an interface "memory.force_reclaim".
> > Any write to this file will drop all charges in this cgroup if
> > there is no task under.
> >
> > %echo 1 > /...../memory.force_reclaim
> >
>
> Looks like a good name, do you think system administrators would
> find force_empty more useful?
>
good name :) I'll use it.

> > +static void
> > +mem_cgroup_force_reclaim_list(struct mem_cgroup *mem, struct list_head *list)
> > +{
> > + struct page_cgroup *pc;
> > + struct page *page;
> > + int count = SWAP_CLUSTER_MAX;
> > + unsigned long flags;
> > +
> > + spin_lock_irqsave(&mem->lru_lock, flags);
> > +
>
> Can we add a comment here stating that this routine reclaims just
> from the per cgroup LRU and not from the zone LRU to which the
> page belongs.
>
Ok.

> > + while (!list_empty(list)) {
> > + pc = list_entry(list->prev, struct page_cgroup, lru);
> > + page = pc->page;
> > + if (clear_page_cgroup(page, pc) == pc) {
> > + css_put(&mem->css);
> > + res_counter_uncharge(&mem->res, PAGE_SIZE);
> > + list_del_init(&pc->lru);
> > + kfree(pc);
> > + } else
> > + count = 1; /* race? ...do relax */
> > +
> > + if (--count == 0) {

```
> > +   spin_unlock_irqrestore(&mem->lru_lock, flags);
> > +   cond_resched();
> > +   spin_lock_irqsave(&mem->lru_lock, flags);
> > +   count = SWAP_CLUSTER_MAX;
> > +  }
> > + }
> > + spin_unlock_irqrestore(&mem->lru_lock, flags);
> > +}
> > +
> > +int mem_cgroup_force_reclaim(struct mem_cgroup *mem)
> > +{
> > + int ret = -EBUSY;
> > + while (!list_empty(&mem->active_list) ||
> > +        !list_empty(&mem->inactive_list)) {
> > +  if (atomic_read(&mem->css.cgroup->count) > 0)
> > +   goto out;
> > +  mem_cgroup_force_reclaim_list(mem, &mem->active_list);
> > +  mem_cgroup_force_reclaim_list(mem, &mem->inactive_list);
> > + }
> > + ret = 0;
> > +out:
> > + css_put(&mem->css);
>
> We do a css_put() here, did we do a css_get() anywhere?
>
Good catch. it is a BUG. I'll fix.

Thanks,
-Kame
```