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

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

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

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

This one's Basic algorithm is

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

A case study.

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

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

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

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

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

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

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

Roll-back can happen when

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

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

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

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

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

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

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

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

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

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

How about you ?

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

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

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

 #include <asm/uaccess.h>

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

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

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

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

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

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

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

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

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

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

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

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

 /*
```

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by yamamoto on Tue, 10 Jun 2008 05:50:32 GMT
View Forum Message <> Reply to Message

> For avoiding complicated rollbacks,
> I think of following ways of policy for task moving (you can add here.)
>
> 1. Before moving usage, reserve usage in the new cgroup and old cgroup.
>   Pros.
>   - rollback will be very easy.
>   Cons.
>   - A task will use twice of its own usage virtaually for a while.
>   - some amount of cpu time will be necessary to move _Big_ apps.
>   - It's difficut to move _Big_ apps to small memcg.
>   - we have to add "special case" handling.
>
> 2. Don't move any usage at task move. (current implementation.)
>   Pros.
>   - no complication in the code.
>   Cons.
>   - A task's usage is chareged to wrong cgroup.
>   - Not sure, but I believe the users don't want this.
>
> 3. Use Lazy Manner
>   When the task moves, we can mark the pages used by it as
>   "Wrong Charge, Should be dropped", and add them some penalty in the LRU.
>   Pros.
>   - no complicated ones.
>   - the pages will be gradually moved at memory pressure.
>   Cons.
>   - A task's usage can exceed the limit for a while.
>   - can't handle mlocked() memory in proper way.
>
> 4. Allow Half-moved state and abandon rollback.
>   Pros.
>   - no complicated ones in the code.
>   Cons.
>   - the users will be in chaos.

how about:

5. try to move charges as your patch does.
   if the target cgroup's usage is going to exceed the limit,
   try to shrink it. if it failed, just leave it exceeded.
   (ie. no rollback)
   for the memory subsystem, which can use its OOM killer,
   the failure should be rare.

> After writing this patch, for me, "3" is attractive. now.
> (or using Lazy manner and allow moving of usage instead of freeing it.)
>
> One reasone is that I think a typical usage of memory controller is

> fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.

i guess that moving long-running applications can be desirable
esp. for not so well-designed systems.

YAMAMOTO Takashi

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Daisuke Nishimura on Tue, 10 Jun 2008 07:35:50 GMT
View Forum Message <> Reply to Message

Hi, Kamezawa-san.

Sorry for late reply.

On Fri, 6 Jun 2008 10:52:35 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
wrote:
> Move Usage at Task Move (just an experimantal for discussion)
> I tested this but don't think bug-free.
>
> In current memcg, when task moves to a new cg, the usage remains in the old cg.
> This is considered to be not good.
>
I agree.

> This is a trial to move "usage" from old cg to new cg at task move.
> Finally, you'll see the problems we have to handle are failure and rollback.
>
> This one's Basic algorithm is
>
> 0. can_attach() is called.
> 1. count movable pages by scanning page table. isolate all pages from LRU.
> 2. try to create enough room in new memory cgroup
> 3. start moving page accouing
> 4. putback pages to LRU.
> 5. can_attach() for other cgroups are called.
>
You isolate pages and move charges of them by can_attach(),
but it means that pages that are allocated between page isolation
and moving tsk->cgroups remains charged to old group, right?

I think it would be better if possible to move charges by attach()
as cpuset migrates pages by cpuset_attach().

But one of the problem of it is that attch() does not return
any value, so there is no way to notify failure...

> A case study.
>
> group_A -> limit=1G, task_X's usage= 800M.
> group_B -> limit=1G, usage=500M.
>
> For moving task_X from group_A to group_B.
> - group_B should be reclaimed or have enough room.
>
> While moving task_X from group_A to group_B.
> - group_B's memory usage can be changed
> - group_A's memory usage can be changed
>
> We accounts the resouce based on pages. Then, we can't move all resource
> usage at once.
>
> If group_B has no more room when we've moved 700M of task_X to group_B,
> we have to move 700M of task_X back to group_A. So I implemented roll-back.
> But other process may use up group_A's available resource at that point.
>
> For avoiding that, preserve 800M in group_B before moving task_X means that
> task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)
>
> This patch uses Best-Effort rollback. Failure in rollback is ignored and
> the usage is just leaked.
>
If implement rollback in kernel, I think it must not fail to prevent
leak of usage.
How about using "charge_force" for rollbak?

Or, instead of implementing rollback in kernel,
how about making user(or middle ware?) re-echo pid to rollbak
on failure?

> Roll-back can happen when
>   (a) in phase 3. cannot move a page to new cgroup because of limit.
>   (b) in phase 5. other cgourp subsys returns error in can_attach().
>
Isn't rollbak needed on failure between can_attach and attach(e.g. failure
on find_css_set, ...)?

> +int mem_cgroup_recharge_task(struct mem_cgroup *newcg,
> +   struct task_struct *task)
> +{
(snip)
> + /* create enough room before move */

> + necessary = info.count * PAGE_SIZE;
> +
> + do {
> + spin_lock(&newcg->res.lock);
> + if (newcg->res.limit > necessary)
> +   rc = -ENOMEM;
I think it should be (newcg->res.limit < necessary).

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Tue, 10 Jun 2008 08:11:26 GMT
View Forum Message <> Reply to Message

On Tue, 10 Jun 2008 14:50:32 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > 3. Use Lazy Manner
> > When the task moves, we can mark the pages used by it as
> > "Wrong Charge, Should be dropped", and add them some penalty in the LRU.
> > Pros.
> > - no complicated ones.
> > - the pages will be gradually moved at memory pressure.
> > Cons.
> > - A task's usage can exceed the limit for a while.
> > - can't handle mlocked() memory in proper way.
> >
> > 4. Allow Half-moved state and abandon rollback.
> > Pros.
> > - no complicated ones in the code.
> > Cons.
> > - the users will be in chaos.
>
> how about:
>
> 5. try to move charges as your patch does.
> if the target cgroup's usage is going to exceed the limit,
> try to shrink it. if it failed, just leave it exceeded.
> (ie. no rollback)
> for the memory subsystem, which can use its OOM killer,
> the failure should be rare.

>

Hmm, allowing exceed and cause OOM kill ?

One difficult point is that the users cannot know they can move task
without any risk. How to handle the risk can be a point.
I don't like that approarch in general because I don't like "exceed"
status. But implementation will be easy.

> > After writing this patch, for me, "3" is attractive. now.
> > (or using Lazy manner and allow moving of usage instead of freeing it.)
> >
> > One reasone is that I think a typical usage of memory controller is
> > fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
>
> i guess that moving long-running applications can be desirable
> esp. for not so well-designed systems.
>

hmm, for not so well-designed systems....true.
But "5" has the same kind of risks for not so well-desgined systems ;)

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Tue, 10 Jun 2008 08:24:41 GMT
View Forum Message <> Reply to Message

On Tue, 10 Jun 2008 16:35:50 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> Hi, Kamezawa-san.
>
> Sorry for late reply.
>
> On Fri, 6 Jun 2008 10:52:35 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
wrote:
> > Move Usage at Task Move (just an experimantal for discussion)
> > I tested this but don't think bug-free.
> >

> > In current memcg, when task moves to a new cg, the usage remains in the old cg.
> > This is considered to be not good.
> >
> I agree.
>
> > This is a trial to move "usage" from old cg to new cg at task move.
> > Finally, you'll see the problems we have to handle are failure and rollback.
> >
> > This one's Basic algorithm is
> >
> > 0. can_attach() is called.
> > 1. count movable pages by scanning page table. isolate all pages from LRU.
> > 2. try to create enough room in new memory cgroup
> > 3. start moving page accouing
> > 4. putback pages to LRU.
> > 5. can_attach() for other cgroups are called.
> >
> You isolate pages and move charges of them by can_attach(),
> but it means that pages that are allocated between page isolation
> and moving tsk->cgroups remains charged to old group, right?
yes.

>
> I think it would be better if possible to move charges by attach()
> as cpuset migrates pages by cpuset_attach().
> But one of the problem of it is that attch() does not return
> any value, so there is no way to notify failure...
>
yes, here again. it makes roll-back more difficult.

> > A case study.
> >
> > group_A -> limit=1G, task_X's usage= 800M.
> > group_B -> limit=1G, usage=500M.
> >
> > For moving task_X from group_A to group_B.
> > - group_B should be reclaimed or have enough room.
> >
> > While moving task_X from group_A to group_B.
> > - group_B's memory usage can be changed
> > - group_A's memory usage can be changed
> >
> > We accounts the resouce based on pages. Then, we can't move all resource
> > usage at once.
> >
> > If group_B has no more room when we've moved 700M of task_X to group_B,
> > we have to move 700M of task_X back to group_A. So I implemented roll-back.
> > But other process may use up group_A's available resource at that point.

> >
> > For avoiding that, preserve 800M in group_B before moving task_X means that
> > task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)
> >
> > This patch uses Best-Effort rollback. Failure in rollback is ignored and
> > the usage is just leaked.
> >
> If implement rollback in kernel, I think it must not fail to prevent
> leak of usage.
> How about using "charge_force" for rollbak?
>
means allowing to exceed limit ?

> Or, instead of implementing rollback in kernel,
> how about making user(or middle ware?) re-echo pid to rollbak
> on failure?
>

"If the users does well, the system works in better way" is O.K.
"If the users doesn't well, the system works in broken way" is very bad.

This is an issue that the kernel should handle by itself.
So this is annoying me.
But we can choice our policy of this task_move. The problem depends
on the policy we establish. So, there will be a good way.
What is "broken" depends on the definition. But usage > limit case
is tend to be considered to be broken.

> > Roll-back can happen when
> >   (a) in phase 3. cannot move a page to new cgroup because of limit.
> >   (b) in phase 5. other cgourp subsys returns error in can_attach().
> >
> Isn't rollbak needed on failure between can_attach and attach(e.g. failure
> on find_css_set, ...)?
>
Yes, my mistake.

But...maybe failure after can_attach() is not good...(for me.)
Paul, how do you think ?
ss->attach() should return a value and fail ?

> > +int mem_cgroup_recharge_task(struct mem_cgroup *newcg,
> > +   struct task_struct *task)
> > +{
> (snip)

> > + /* create enough room before move */
> > + necessary = info.count * PAGE_SIZE;
> > +
> > + do {
> > + spin_lock(&newcg->res.lock);
> > + if (newcg->res.limit > necessary)
> > +   rc = -ENOMEM;
> I think it should be (newcg->res.limit < necessary).
>
Ah, you're right. should be fixed.

Anyway I'll rewrite the whole considering opions from others.

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by yamamoto on Tue, 10 Jun 2008 12:57:03 GMT
View Forum Message <> Reply to Message

> On Tue, 10 Jun 2008 14:50:32 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > > 3. Use Lazy Manner
> > > When the task moves, we can mark the pages used by it as
> > > "Wrong Charge, Should be dropped", and add them some penalty in the LRU.
> > > Pros.
> > > - no complicated ones.
> > > - the pages will be gradually moved at memory pressure.
> > > Cons.
> > > - A task's usage can exceed the limit for a while.
> > > - can't handle mlocked() memory in proper way.
> > >
> > > 4. Allow Half-moved state and abandon rollback.
> > > Pros.
> > > - no complicated ones in the code.
> > > Cons.
> > > - the users will be in chaos.
> >
> > how about:
> >
> > 5. try to move charges as your patch does.

> > if the target cgroup's usage is going to exceed the limit,
> > try to shrink it. if it failed, just leave it exceeded.
> > (ie. no rollback)
> > for the memory subsystem, which can use its OOM killer,
> > the failure should be rare.
> >
>
> Hmm, allowing exceed and cause OOM kill ?
>
> One difficult point is that the users cannot know they can move task
> without any risk. How to handle the risk can be a point.
> I don't like that approarch in general because I don't like "exceed"
> status. But implementation will be easy.

regardless of how to handle task moves,
it's important to provide information to help users
to avoid unreasonable cgroup/task placement.
otherwise, they will be surprised by OOM-killer etc anyway.

having said that, if you decide to put too large tasks into
a cgroup with too small limit, i don't think that there are
many choices besides OOM-kill and allowing "exceed".

actually, i think that #3 and #5 are somewhat similar.
a big difference is that, while #5 shrinks the cgroup immediately,
#3 does it later. in case we need to do OOM-kill, i prefer to do it
sooner than later.

> > > After writing this patch, for me, "3" is attractive. now.
> > > (or using Lazy manner and allow moving of usage instead of freeing it.)
> > >
> > > One reasone is that I think a typical usage of memory controller is
> > > fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
> >
> > i guess that moving long-running applications can be desirable
> > esp. for not so well-designed systems.
> >
>
> hmm, for not so well-designed systems....true.
> But "5" has the same kind of risks for not so well-desgined systems ;)

i don't claim that #5 is a perfect solution for everyone. :)

YAMAMOTO Takashi

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 01:58:41 GMT
View Forum Message <> Reply to Message

On Tue, 10 Jun 2008 21:57:03 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> > > 5. try to move charges as your patch does.
> > > if the target cgroup's usage is going to exceed the limit,
> > > try to shrink it. if it failed, just leave it exceeded.
> > > (ie. no rollback)
> > > for the memory subsystem, which can use its OOM killer,
> > > the failure should be rare.
> > >
> >
> > Hmm, allowing exceed and cause OOM kill ?
> >
> > One difficult point is that the users cannot know they can move task
> > without any risk. How to handle the risk can be a point.
> > I don't like that approarch in general because I don't like "exceed"
> > status. But implementation will be easy.
>
> regardless of how to handle task moves,
> it's important to provide information to help users
> to avoid unreasonable cgroup/task placement.
> otherwise, they will be surprised by OOM-killer etc anyway.
>
yes.

> having said that, if you decide to put too large tasks into
> a cgroup with too small limit, i don't think that there are
> many choices besides OOM-kill and allowing "exceed".
>
IMHO, allowing exceed is harmfull without changing the definition of "limit".
"limit" is hard-limit, now, not soft-limit. Changing the defintion just for
this is not acceptable for me.
Maybe "move" under limit itself is crazy ops....Hmm...

Should we allow task move when the destination cgroup is unlimited ?
Isn't it useful ?

> actually, i think that #3 and #5 are somewhat similar.
> a big difference is that, while #5 shrinks the cgroup immediately,
> #3 does it later. in case we need to do OOM-kill, i prefer to do it
> sooner than later.
>
#3 will not cause OOM-killer, I hope...A user can notice memory shortage.

> > > > After writing this patch, for me, "3" is attractive. now.
> > > > (or using Lazy manner and allow moving of usage instead of freeing it.)
> > > >
> > > > One reasone is that I think a typical usage of memory controller is
> > > > fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
> > >
> > > i guess that moving long-running applications can be desirable
> > > esp. for not so well-designed systems.
> > >
> >
> > hmm, for not so well-designed systems....true.
> > But "5" has the same kind of risks for not so well-desgined systems ;)
>
> i don't claim that #5 is a perfect solution for everyone. :)
>

Maybe there will no perfect solution ;)

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Daisuke Nishimura on Wed, 11 Jun 2008 03:03:45 GMT
View Forum Message <> Reply to Message

On Tue, 10 Jun 2008 17:26:37 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > > This is a trial to move "usage" from old cg to new cg at task move.
> > > Finally, you'll see the problems we have to handle are failure and rollback.
> > >
> > > This one's Basic algorithm is
> > >
> > > 0. can_attach() is called.
> > > 1. count movable pages by scanning page table. isolate all pages from LRU.
> > > 2. try to create enough room in new memory cgroup
> > > 3. start moving page accouing
> > > 4. putback pages to LRU.
> > > 5. can_attach() for other cgroups are called.
> > >
> > You isolate pages and move charges of them by can_attach(),

> > but it means that pages that are allocated between page isolation
> > and moving tsk->cgroups remains charged to old group, right?
> yes.
>
> >
> > I think it would be better if possible to move charges by attach()
> > as cpuset migrates pages by cpuset_attach().
> > But one of the problem of it is that attch() does not return
> > any value, so there is no way to notify failure...
> >
> yes, here again. it makes roll-back more difficult.
>
I think so too. That's why I said "one of the problem".

> > > A case study.
> > >
> > > group_A -> limit=1G, task_X's usage= 800M.
> > > group_B -> limit=1G, usage=500M.
> > >
> > > For moving task_X from group_A to group_B.
> > > - group_B should be reclaimed or have enough room.
> > >
> > > While moving task_X from group_A to group_B.
> > > - group_B's memory usage can be changed
> > > - group_A's memory usage can be changed
> > >
> > > We accounts the resouce based on pages. Then, we can't move all resource
> > > usage at once.
> > >
> > > If group_B has no more room when we've moved 700M of task_X to group_B,
> > > we have to move 700M of task_X back to group_A. So I implemented roll-back.
> > > But other process may use up group_A's available resource at that point.
> > >
> > > For avoiding that, preserve 800M in group_B before moving task_X means that
> > > task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)
> > >
> > > This patch uses Best-Effort rollback. Failure in rollback is ignored and
> > > the usage is just leaked.
> > >
> > If implement rollback in kernel, I think it must not fail to prevent
> > leak of usage.
> > How about using "charge_force" for rollbak?
> >
> means allowing to exceed limit ?
>
Yes.
I agree that exceeding limit is not good, but I
just feel that it's better than leaking usage.

Of cource, I think usage should be decreased later
by some methods.

> > Or, instead of implementing rollback in kernel,
> > how about making user(or middle ware?) re-echo pid to rollbak
> > on failure?
> >
>
> "If the users does well, the system works in better way" is O.K.
> "If the users doesn't well, the system works in broken way" is very bad.
>
Hum...

I think users must know what they are doing.

They must know that moving a process to another group
that doesn't have enough room for it may fail with half state,
if it is the behavior of kernel.
And they should handle the error by themselves, IMHO.

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 03:24:56 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 12:03:45 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
> > > Or, instead of implementing rollback in kernel,
> > > how about making user(or middle ware?) re-echo pid to rollbak
> > > on failure?
> > >
> >
> > "If the users does well, the system works in better way" is O.K.
> > "If the users doesn't well, the system works in broken way" is very bad.
> >
> Hum...
>
> I think users must know what they are doing.
>
yes. but it's a different problem,

 - "a user must know what they does."
 - "a system works without BUG even if the user is crazy."

> They must know that moving a process to another group
> that doesn't have enough room for it may fail with half state,
> if it is the behavior of kernel.
> And they should handle the error by themselves, IMHO.
>

I'm now considering following logic. How do you think ?

Assume: move TASK from group:CURR to group:DEST.

== move_task(TASK, CURR, DEST)

if (DEST's limit is unlimited)
 moving TASK
 return success.

usage = check_usage_of_task(TASK).

/* try to reserve enough room in destionation */
if (try_to_reserve_enough_room(DEST, usage)) {
 move TASK to DEST and move pages AMAP.
 /* usage_of_task(TASK) can be changed while we do this.
    Then, we move AMAP. */
 return success;
}
return failure.
==

The difficult point will be reservation but can be implemented without
complexity.

Thanks,
-Kame

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by yamamoto on Wed, 11 Jun 2008 03:44:46 GMT
View Forum Message <> Reply to Message

> I'm now considering following logic. How do you think ?
>
> Assume: move TASK from group:CURR to group:DEST.
>
> == move_task(TASK, CURR, DEST)
>
> if (DEST's limit is unlimited)
> moving TASK
> return success.
>
> usage = check_usage_of_task(TASK).
>
> /* try to reserve enough room in destionation */
> if (try_to_reserve_enough_room(DEST, usage)) {
> move TASK to DEST and move pages AMAP.
> /* usage_of_task(TASK) can be changed while we do this.
> Then, we move AMAP. */
> return success;
> }
> return failure.
> ==

AMAP means that you might leave some random charges in CURR?

i think that you can redirect new charges in TASK to DEST
so that usage_of_task(TASK) will not grow.

YAMAMOTO Takashi

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by yamamoto on Wed, 11 Jun 2008 03:45:14 GMT
View Forum Message <> Reply to Message

> > having said that, if you decide to put too large tasks into
> > a cgroup with too small limit, i don't think that there are

> > many choices besides OOM-kill and allowing "exceed".
> >
> IMHO, allowing exceed is harmfull without changing the definition of "limit".
> "limit" is hard-limit, now, not soft-limit. Changing the defintion just for
> this is not acceptable for me.

even with the current code, the "exceed" condition can be created
by simply lowering the limit.
(well, i know that some of your patches floating around change it.)

> Maybe "move" under limit itself is crazy ops....Hmm...
>
> Should we allow task move when the destination cgroup is unlimited ?
> Isn't it useful ?

i think it makes some sense.

> > actually, i think that #3 and #5 are somewhat similar.
> > a big difference is that, while #5 shrinks the cgroup immediately,
> > #3 does it later. in case we need to do OOM-kill, i prefer to do it
> > sooner than later.
> >
> #3 will not cause OOM-killer, I hope...A user can notice memory shortage.

we are talking about the case where a cgroup's working set is getting
hopelessly larger than its limit. i don't see why #3 will not
cause OOM-kill. can you explain?

YAMAMOTO Takashi

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 04:05:32 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 12:45:14 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > > having said that, if you decide to put too large tasks into
> > > a cgroup with too small limit, i don't think that there are
> > > many choices besides OOM-kill and allowing "exceed".
> > >
> > IMHO, allowing exceed is harmfull without changing the definition of "limit".
> > "limit" is hard-limit, now, not soft-limit. Changing the defintion just for

> > this is not acceptable for me.
>
> even with the current code, the "exceed" condition can be created
> by simply lowering the limit.
> (well, i know that some of your patches floating around change it.)
>
Yes, I write it now ;) Handling exceed contains some troubles

 - when resizing limit, to what extent exceed is allowed ?
 - Once exceed, no new page allocation can success and
   some random process will die because of OOM.

> > Maybe "move" under limit itself is crazy ops....Hmm...
> >
> > Should we allow task move when the destination cgroup is unlimited ?
> > Isn't it useful ?
>
> i think it makes some sense.
>
> > > actually, i think that #3 and #5 are somewhat similar.
> > > a big difference is that, while #5 shrinks the cgroup immediately,
> > > #3 does it later. in case we need to do OOM-kill, i prefer to do it
> > > sooner than later.
> > >
> > #3 will not cause OOM-killer, I hope...A user can notice memory shortage.
>
> we are talking about the case where a cgroup's working set is getting
> hopelessly larger than its limit. i don't see why #3 will not
> cause OOM-kill. can you explain?
>
just because #3 doesn't move resource, just drop.

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 04:14:37 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 12:44:46 +0900 (JST)

yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > I'm now considering following logic. How do you think ?
> >
> > Assume: move TASK from group:CURR to group:DEST.
> >
> > == move_task(TASK, CURR, DEST)
> >
> > if (DEST's limit is unlimited)
> > moving TASK
> > return success.
> >
> > usage = check_usage_of_task(TASK).
> >
> > /* try to reserve enough room in destionation */
> > if (try_to_reserve_enough_room(DEST, usage)) {
> > move TASK to DEST and move pages AMAP.
> > /* usage_of_task(TASK) can be changed while we do this.
> > Then, we move AMAP. */
> > return success;
> > }
> > return failure.
> > ==
>
> AMAP means that you might leave some random charges in CURR?
>
yes. but we can reduce bad case by some way
 - reserve more than necessary.
 or
 - read_lock mm->sem while move.

> i think that you can redirect new charges in TASK to DEST
> so that usage_of_task(TASK) will not grow.
>

Hmm, to do that, we have to handle complicated cgroup's attach ops.

at this moving, memcg is pointed by
 - TASK->cgroup->memcg(CURR)
after move
 - TASK->another_cgroup->memcg(DEST)

This move happens before cgroup is replaced by another_cgroup.

Thanks,
-Kame

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Daisuke Nishimura on Wed, 11 Jun 2008 04:29:09 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 13:14:37 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> On Wed, 11 Jun 2008 12:44:46 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > > I'm now considering following logic. How do you think ?
> > >
> > > Assume: move TASK from group:CURR to group:DEST.
> > >
> > > == move_task(TASK, CURR, DEST)
> > >
> > > if (DEST's limit is unlimited)
> > > moving TASK
> > > return success.
> > >
> > > usage = check_usage_of_task(TASK).
> > >
> > > /* try to reserve enough room in destionation */
> > > if (try_to_reserve_enough_room(DEST, usage)) {
> > > move TASK to DEST and move pages AMAP.
> > > /* usage_of_task(TASK) can be changed while we do this.
> > > Then, we move AMAP. */
> > > return success;
> > > }
> > > return failure.
> > > ==
> >
> > AMAP means that you might leave some random charges in CURR?
> >
> yes. but we can reduce bad case by some way
> - reserve more than necessary.
> or
> - read_lock mm->sem while move.
>
I preffer the latter.
Though it's expencive, I think moving a task would not happen
so offen.

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 04:37:10 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 13:29:09 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> On Wed, 11 Jun 2008 13:14:37 +0900, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > On Wed, 11 Jun 2008 12:44:46 +0900 (JST)
> > yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> >
> > > > I'm now considering following logic. How do you think ?
> > > >
> > > > Assume: move TASK from group:CURR to group:DEST.
> > > >
> > > > == move_task(TASK, CURR, DEST)
> > > >
> > > > if (DEST's limit is unlimited)
> > > > moving TASK
> > > > return success.
> > > >
> > > > usage = check_usage_of_task(TASK).
> > > >
> > > > /* try to reserve enough room in destionation */
> > > > if (try_to_reserve_enough_room(DEST, usage)) {
> > > > move TASK to DEST and move pages AMAP.
> > > > /* usage_of_task(TASK) can be changed while we do this.
> > > > Then, we move AMAP. */
> > > > return success;
> > > > }
> > > > return failure.
> > > > ==
> > >
> > > AMAP means that you might leave some random charges in CURR?
> > >
> > yes. but we can reduce bad case by some way
> > - reserve more than necessary.
> > or

> > - read_lock mm->sem while move.
> >
> I preffer the latter.
> Though it's expencive, I think moving a task would not happen
> so offen.
>
Sure.

I'd like to write one and post as RFC. (hopefully in this week)

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Paul Menage on Wed, 11 Jun 2008 07:17:31 GMT
View Forum Message <> Reply to Message

On Thu, Jun 5, 2008 at 6:52 PM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> Move Usage at Task Move (just an experimantal for discussion)
> I tested this but don't think bug-free.
>
> In current memcg, when task moves to a new cg, the usage remains in the old cg.
> This is considered to be not good.

Is it really such a big deal if we don't transfer the page ownerships
to the new cgroup? As this thread has shown, it's a fairly painful
operation to support. It would be good to have some concrete examples
of cases where this is needed.

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

>   0. can_attach() is called.
>   1. count movable pages by scanning page table. isolate all pages from LRU.
>   2. try to create enough room in new memory cgroup
>   3. start moving page accouing
>   4. putback pages to LRU.
>   5. can_attach() for other cgroups are called.
>
> A case study.
>
> group_A -> limit=1G, task_X's usage= 800M.
> group_B -> limit=1G, usage=500M.
>
> For moving task_X from group_A to group_B.
> - group_B should be reclaimed or have enough room.
>
> While moving task_X from group_A to group_B.
> - group_B's memory usage can be changed
> - group_A's memory usage can be changed
>
> We accounts the resouce based on pages. Then, we can't move all resource
> usage at once.
>
> If group_B has no more room when we've moved 700M of task_X to group_B,
> we have to move 700M of task_X back to group_A. So I implemented roll-back.
> But other process may use up group_A's available resource at that point.
>
> For avoiding that, preserve 800M in group_B before moving task_X means that
> task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)

I think that pre-reserving in B would be the cleanest solution, and
would save the need to provide rollback.

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

I'd say stick with this unless there a strong arguments in favour of
changing, based on concrete needs.

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

Exactly - this is a good reason *not* to implement move - because then
you drag all the usage of the middleware daemon into the new cgroup.

```
> Index: temp-2.6.26-rc2-mm1/include/linux/cgroup.h
> ===================================================================
> --- temp-2.6.26-rc2-mm1.orig/include/linux/cgroup.h
> +++ temp-2.6.26-rc2-mm1/include/linux/cgroup.h
> @@ -299,6 +299,8 @@ struct cgroup_subsys {
>                       struct cgroup *cgrp, struct task_struct *tsk);
>       void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
>                       struct cgroup *old_cgrp, struct task_struct *tsk);
> +      void (*attach_rollback)(struct cgroup_subsys *ss,
> +                       struct task_struct *tsk);
>       void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
>       void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
>       int (*populate)(struct cgroup_subsys *ss,
> Index: temp-2.6.26-rc2-mm1/kernel/cgroup.c
> ===================================================================
> --- temp-2.6.26-rc2-mm1.orig/kernel/cgroup.c
> +++ temp-2.6.26-rc2-mm1/kernel/cgroup.c
> @@ -1241,7 +1241,7 @@ int cgroup_attach_task(struct cgroup *cg
>               if (ss->can_attach) {
>                       retval = ss->can_attach(ss, cgrp, tsk);
>                       if (retval)
> -                             return retval;
> +                             goto rollback;
>               }
>       }
>
> @@ -1278,6 +1278,13 @@ int cgroup_attach_task(struct cgroup *cg
>       synchronize_rcu();
>       put_css_set(cg);
>       return 0;
> +
> +rollback:
> +      for_each_subsys(root, ss) {
> +              if (ss->attach_rollback)
> +                      ss->attach_rollback(ss, tsk);
> +      }
> +      return retval;
>  }
>
```

I really need to get round to my plan for implementing transactional attach - I've just been swamped by internal stuff recently. Essentially, I think that we need the ability for a subsystem to request either a commit or a rollback following an attach. The big difference to what we have now is that the each subsystem will be able to synchronize itself with the updates to its state pointer in the task's css_set. Also, we need to not be calling attach_rollback on

subsystems that didn't get an attach() call.

Paul

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 07:42:12 GMT
View Forum Message <> Reply to Message

Hi,

On Wed, 11 Jun 2008 00:17:31 -0700
"Paul Menage" <menage@google.com> wrote:

> On Thu, Jun 5, 2008 at 6:52 PM, KAMEZAWA Hiroyuki
> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > Move Usage at Task Move (just an experimantal for discussion)
> > I tested this but don't think bug-free.
> >
> > In current memcg, when task moves to a new cg, the usage remains in the old cg.
> > This is considered to be not good.
>
> Is it really such a big deal if we don't transfer the page ownerships
> to the new cgroup? As this thread has shown, it's a fairly painful
> operation to support. It would be good to have some concrete examples
> of cases where this is needed.
>
When we moves a process with XXXG bytes of memory, we need "move" obviously.

I think there is a case that system administrator decides to create _new_
cgroup to isolate some swappy job for maintaining the system.
(I never be able to say that never happens.)

This kind of resource resizing can be happen under automatic controlls of
middleware, I think. But as you say, this should be implemented in simple way.
I'm now trying to make this simple. (i.e. searching no-rollback approach.)

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

> > 0. can_attach() is called.
> > 1. count movable pages by scanning page table. isolate all pages from LRU.
> > 2. try to create enough room in new memory cgroup
> > 3. start moving page accouing
> > 4. putback pages to LRU.
> > 5. can_attach() for other cgroups are called.
> >
> > A case study.
> >
> > group_A -> limit=1G, task_X's usage= 800M.
> > group_B -> limit=1G, usage=500M.
> >
> > For moving task_X from group_A to group_B.
> > - group_B should be reclaimed or have enough room.
> >
> > While moving task_X from group_A to group_B.
> > - group_B's memory usage can be changed
> > - group_A's memory usage can be changed
> >
> > We accounts the resouce based on pages. Then, we can't move all resource
> > usage at once.
> >
> > If group_B has no more room when we've moved 700M of task_X to group_B,
> > we have to move 700M of task_X back to group_A. So I implemented roll-back.
> > But other process may use up group_A's available resource at that point.
> >
> > For avoiding that, preserve 800M in group_B before moving task_X means that
> > task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)
>
> I think that pre-reserving in B would be the cleanest solution, and
> would save the need to provide rollback.
>
Yes. My next version will try to pre-reserve. and no rollbacks.

> > 2. Don't move any usage at task move. (current implementation.)
> > Pros.
> > - no complication in the code.
> > Cons.
> > - A task's usage is chareged to wrong cgroup.
> > - Not sure, but I believe the users don't want this.
>
> I'd say stick with this unless there a strong arguments in favour of
> changing, based on concrete needs.
>
People around me says "this logic is buggy" ;)

> >
> > One reasone is that I think a typical usage of memory controller is
> > fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
>
> Exactly - this is a good reason *not* to implement move - because then
> you drag all the usage of the middleware daemon into the new cgroup.
>
Yes but this is one of the usage of cgroup. In general, system admin can
use this for limiting memory on his own decision.

> > Index: temp-2.6.26-rc2-mm1/include/linux/cgroup.h
> > ===
> > --- temp-2.6.26-rc2-mm1.orig/include/linux/cgroup.h
> > +++ temp-2.6.26-rc2-mm1/include/linux/cgroup.h
> > @@ -299,6 +299,8 @@ struct cgroup_subsys {
> >   struct cgroup *cgrp, struct task_struct *tsk);
> >   void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
> >   struct cgroup *old_cgrp, struct task_struct *tsk);
> > +   void (*attach_rollback)(struct cgroup_subsys *ss,
> > +   struct task_struct *tsk);
> >   void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
> >   void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
> >   int (*populate)(struct cgroup_subsys *ss,
> > Index: temp-2.6.26-rc2-mm1/kernel/cgroup.c
> > ===
> > --- temp-2.6.26-rc2-mm1.orig/kernel/cgroup.c
> > +++ temp-2.6.26-rc2-mm1/kernel/cgroup.c
> > @@ -1241,7 +1241,7 @@ int cgroup_attach_task(struct cgroup *cg
> >   if (ss->can_attach) {
> >   retval = ss->can_attach(ss, cgrp, tsk);
> >   if (retval)
> > -   return retval;
> > +   goto rollback;
> >   }
> >   }
> >
> > @@ -1278,6 +1278,13 @@ int cgroup_attach_task(struct cgroup *cg
> >   synchronize_rcu();
> >   put_css_set(cg);
> >   return 0;
> > +
> > +rollback:
> > +   for_each_subsys(root, ss) {
> > +   if (ss->attach_rollback)
> > +   ss->attach_rollback(ss, tsk);

> > +	}
> > +	return retval;
> > }
> >
>
> I really need to get round to my plan for implementing transactional
> attach - I've just been swamped by internal stuff recently.
> Essentially, I think that we need the ability for a subsystem to
> request either a commit or a rollback following an attach. The big
> difference to what we have now is that the each subsystem will be able
> to synchronize itself with the updates to its state pointer in the
> task's css_set. Also, we need to not be calling attach_rollback on
> subsystems that didn't get an attach() call.
>
yes. but, at first, I'll try no-rollback approach.
And can I move memory resource controller's subsys_id to the last for now ?

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Paul Menage on Wed, 11 Jun 2008 08:04:14 GMT
View Forum Message <> Reply to Message

On Wed, Jun 11, 2008 at 12:45 AM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
>> Is it really such a big deal if we don't transfer the page ownerships
>> to the new cgroup? As this thread has shown, it's a fairly painful
>> operation to support. It would be good to have some concrete examples
>> of cases where this is needed.
>>
> When we moves a process with XXXG bytes of memory, we need "move" obviously.

That's not a concrete example, it's an assertion :-)

>
> I think there is a case that system administrator decides to create _new_
> cgroup to isolate some swappy job for maintaining the system.
> (I never be able to say that never happens.)

OK, that seems like a reasonable case - i.e. when an existing cgroup

is deliberately split into two.

An alternative way to support that would be to do nothing at move
time, but provide a "pull_usage" control file that would slurp any
pages in any mm in the cgroup into the cgroup.
>> >
>> > One reasone is that I think a typical usage of memory controller is
>> > fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
>>
>> Exactly - this is a good reason *not* to implement move - because then
>> you drag all the usage of the middleware daemon into the new cgroup.
>>
> Yes but this is one of the usage of cgroup. In general, system admin can
> use this for limiting memory on his own decision.
>

Sorry, your last sentence doesn't make sense to me in this context.

If the common mode for middleware starting a new cgroup is fork() /
move / exec() then after the fork(), the child will be sharing pages
with the main daemon process. So the move will pull all the daemon's
memory into the new cgroup

> yes. but, at first, I'll try no-rollback approach.
> And can I move memory resource controller's subsys_id to the last for now ?
>

That's probably fine for experimentation, but it wouldn't be something
we'd want to commit to -mm or mainline.

Paul

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Wed, 11 Jun 2008 08:26:37 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 01:04:14 -0700
"Paul Menage" <menage@google.com> wrote:

> An alternative way to support that would be to do nothing at move
> time, but provide a "pull_usage" control file that would slurp any
> pages in any mm in the cgroup into the cgroup.
> >> >

> >> > One reasone is that I think a typical usage of memory controller is
> >> > fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
> >>
> >> Exactly - this is a good reason *not* to implement move - because then
> >> you drag all the usage of the middleware daemon into the new cgroup.
> >>
> > Yes but this is one of the usage of cgroup. In general, system admin can
> > use this for limiting memory on his own decision.
> >
>
> Sorry, your last sentence doesn't make sense to me in this context.
>
Sorry. try another sentense..

I think cgroup itself is designed to be able to be used without middleware.
IOW, whether using middleware or not is the matter of users not of developpers.
There will be a system that system admin controlles all and move tasks by hand.
ex)...personal notebooks etc..

> If the common mode for middleware starting a new cgroup is fork() /
> move / exec() then after the fork(), the child will be sharing pages
> with the main daemon process. So the move will pull all the daemon's
> memory into the new cgroup
>
My patch (this patch) just moves Private Anon page to new cgroup. (of mapcount=1)

> > yes. but, at first, I'll try no-rollback approach.
> > And can I move memory resource controller's subsys_id to the last for now ?
> >
>
> That's probably fine for experimentation, but it wouldn't be something
> we'd want to commit to -mm or mainline.
>

Hmm, I'd like to post a patch to add "rollback" to cgroup if I find it necessary.
My first purpose of this post is showing the problem and starting discussion.
Anyway, I will remove "RFC" only when I got enough number of Acks.

Thanks,
-Kame

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Balbir Singh on Wed, 11 Jun 2008 08:27:34 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Thu, Jun 5, 2008 at 6:52 PM, KAMEZAWA Hiroyuki
> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>> Move Usage at Task Move (just an experimantal for discussion)
>> I tested this but don't think bug-free.
>>
>> In current memcg, when task moves to a new cg, the usage remains in the old cg.
>> This is considered to be not good.
>
> Is it really such a big deal if we don't transfer the page ownerships
> to the new cgroup? As this thread has shown, it's a fairly painful
> operation to support. It would be good to have some concrete examples
> of cases where this is needed.
>
>

I tend to agree with Paul. One of the reasons, I did not move charges is that
makes migration an expensive operation. Since migration is well controlled with
permissions, we assume that the node owner what he/she is doing.

>> This is a trial to move "usage" from old cg to new cg at task move.
>> Finally, you'll see the problems we have to handle are failure and rollback.
>>
>> This one's Basic algorithm is
>>
>>   0. can_attach() is called.
>>   1. count movable pages by scanning page table. isolate all pages from LRU.
>>   2. try to create enough room in new memory cgroup
>>   3. start moving page accouing
>>   4. putback pages to LRU.
>>   5. can_attach() for other cgroups are called.
>>
>> A case study.
>>
>> group_A -> limit=1G, task_X's usage= 800M.
>> group_B -> limit=1G, usage=500M.
>>
>> For moving task_X from group_A to group_B.
>> - group_B should be reclaimed or have enough room.
>>
>> While moving task_X from group_A to group_B.
>> - group_B's memory usage can be changed
>> - group_A's memory usage can be changed
>>
>> We accounts the resouce based on pages. Then, we can't move all resource

>> usage at once.
>>
>> If group_B has no more room when we've moved 700M of task_X to group_B,
>> we have to move 700M of task_X back to group_A. So I implemented roll-back.
>> But other process may use up group_A's available resource at that point.
>>
>> For avoiding that, preserve 800M in group_B before moving task_X means that
>> task_X can occupy 1600M of resource at moving. (So I don't do in this patch.)
>
> I think that pre-reserving in B would be the cleanest solution, and
> would save the need to provide rollback.
>
>> 2. Don't move any usage at task move. (current implementation.)
>>   Pros.
>>   - no complication in the code.
>>   Cons.
>>   - A task's usage is chareged to wrong cgroup.
>>   - Not sure, but I believe the users don't want this.
>
> I'd say stick with this unless there a strong arguments in favour of
> changing, based on concrete needs.
>
>> One reasone is that I think a typical usage of memory controller is
>> fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
>
> Exactly - this is a good reason *not* to implement move - because then
> you drag all the usage of the middleware daemon into the new cgroup.
>

Yes. The other thing is that charges will eventually fade away. Please see the
cgroup implementation of page_referenced() and mark_page_accessed(). The
original group on memory pressure will drop pages that were left behind by a
task that migrates. The new group will pick it up if referenced.

[snip]

--
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: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Paul Menage on Wed, 11 Jun 2008 08:48:20 GMT
View Forum Message <> Reply to Message

On Wed, Jun 11, 2008 at 1:27 AM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> Sorry. try another sentense..
>
> I think cgroup itself is designed to be able to be used without middleware.

True, but it shouldn't be hostile to middleware, since I think that
automated use will be much more common. (And certainly if you count
the number of servers :-))

> IOW, whether using middleware or not is the matter of users not of developpers.
> There will be a system that system admin controlles all and move tasks by hand.
> ex)...personal notebooks etc..
>

You think so? I think that at the very least users will be using tools
based around config scripts, rule engines and libcgroup, if not a
persistent daemon.

>> If the common mode for middleware starting a new cgroup is fork() /
>> move / exec() then after the fork(), the child will be sharing pages
>> with the main daemon process. So the move will pull all the daemon's
>> memory into the new cgroup
>>
> My patch (this patch) just moves Private Anon page to new cgroup. (of mapcount=1)

OK, well that makes it more reasonable regarding the above problem.
But I can still see problems if, say, a single thread moves into a new
cgroup, you move the entire memory. Perhaps you should only do so if
the mm->owner changes task?

Paul

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Daisuke Nishimura on Wed, 11 Jun 2008 12:21:26 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 13:57:34 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

(snip)

> >> 2. Don't move any usage at task move. (current implementation.)
> >>   Pros.
> >>   - no complication in the code.
> >>   Cons.
> >>   - A task's usage is chareged to wrong cgroup.
> >>   - Not sure, but I believe the users don't want this.
> >
> > I'd say stick with this unless there a strong arguments in favour of
> > changing, based on concrete needs.
> >
> >> One reasone is that I think a typical usage of memory controller is
> >> fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
> >
> > Exactly - this is a good reason *not* to implement move - because then
> > you drag all the usage of the middleware daemon into the new cgroup.
> >
>
> Yes. The other thing is that charges will eventually fade away. Please see the
> cgroup implementation of page_referenced() and mark_page_accessed(). The
> original group on memory pressure will drop pages that were left behind by a
> task that migrates. The new group will pick it up if referenced.
>
Hum..
So, it seems that some kind of "Lazy Mode"(#3 of Kamezawa-san's)
has been implemented already.

But, one of the reason that I think usage should be moved
is to make the usage as accurate as possible, that is
the size of memory used by processes in the group at the moment.

I agree that statistics is not the purpose of memcg(and swap),
but, IMHO, it's useful feature of memcg.
Administrators can know how busy or idle each groups are by it.

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: Re: [RFD][PATCH] memcg: Move Usage at Task Move

----- Original Message -----
>On Wed, 11 Jun 2008 13:57:34 +0530
>Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
>(snip)
>
>> >> 2. Don't move any usage at task move. (current implementation.)
>> >>   Pros.
>> >>   - no complication in the code.
>> >>   Cons.
>> >>   - A task's usage is chareged to wrong cgroup.
>> >>   - Not sure, but I believe the users don't want this.
>> >
>> > I'd say stick with this unless there a strong arguments in favour of
>> > changing, based on concrete needs.
>> >
>> >> One reasone is that I think a typical usage of memory controller is
>> >> fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
>> >
>> > Exactly - this is a good reason *not* to implement move - because then
>> > you drag all the usage of the middleware daemon into the new cgroup.
>> >
>>
>> Yes. The other thing is that charges will eventually fade away. Please see
the
>> cgroup implementation of page_referenced() and mark_page_accessed(). The
>> original group on memory pressure will drop pages that were left behind by
a
>> task that migrates. The new group will pick it up if referenced.
>>
>Hum..
>So, it seems that some kind of "Lazy Mode"(#3 of Kamezawa-san's)
>has been implemented already.
>
>But, one of the reason that I think usage should be moved
>is to make the usage as accurate as possible, that is
>the size of memory used by processes in the group at the moment.
>
>I agree that statistics is not the purpose of memcg(and swap),
>but, IMHO, it's useful feature of memcg.
>Administrators can know how busy or idle each groups are by it.
>
One more point. This kinds of lazy "drop" approach canoot works well when
there are mlocked processes. lazy "move" approarch is better if we do in lazy
way. And how quickly they drops depends on vm.swappiness.

Anyway, I don't like complicated logic in the kernel.
So, let's see how simple "move" can be implemented. Then, it will be just a
trade-off problem, IMHO.
If policy is fixed, implementation itself will not be complicated, I think.

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by Balbir Singh on Wed, 11 Jun 2008 13:13:38 GMT
View Forum Message <> Reply to Message

kamezawa.hiroyu@jp.fujitsu.com wrote:
> ----- Original Message -----
>> On Wed, 11 Jun 2008 13:57:34 +0530
>> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>>
>> (snip)
>>
>>>>> 2. Don't move any usage at task move. (current implementation.)
>>>>>   Pros.
>>>>>   - no complication in the code.
>>>>>   Cons.
>>>>>   - A task's usage is chareged to wrong cgroup.
>>>>>   - Not sure, but I believe the users don't want this.
>>>> I'd say stick with this unless there a strong arguments in favour of
>>>> changing, based on concrete needs.
>>>>
>>>>> One reasone is that I think a typical usage of memory controller is
>>>>> fork()->move->exec(). (by libcg ?) and exec() will flush the all usage.
>>>> Exactly - this is a good reason *not* to implement move - because then
>>>> you drag all the usage of the middleware daemon into the new cgroup.
>>>>
>>> Yes. The other thing is that charges will eventually fade away. Please see
> the
>>> cgroup implementation of page_referenced() and mark_page_accessed(). The
>>> original group on memory pressure will drop pages that were left behind by
> a
>>> task that migrates. The new group will pick it up if referenced.
>>>
>> Hum..
>> So, it seems that some kind of "Lazy Mode"(#3 of Kamezawa-san's)

>> has been implemented already.
>>
>> But, one of the reason that I think usage should be moved
>> is to make the usage as accurate as possible, that is
>> the size of memory used by processes in the group at the moment.
>>
>> I agree that statistics is not the purpose of memcg(and swap),
>> but, IMHO, it's useful feature of memcg.
>> Administrators can know how busy or idle each groups are by it.
>>
> One more point. This kinds of lazy "drop" approach canoot works well when
> there are mlocked processes. lazy "move" approarch is better if we do in lazy
> way. And how quickly they drops depends on vm.swappiness.
>
> Anyway, I don't like complicated logic in the kernel.
> So, let's see how simple "move" can be implemented. Then, it will be just a
> trade-off problem, IMHO.
> If policy is fixed, implementation itself will not be complicated, I think.
>

I agree with you that it is a trade-off problem and we should keep move as
simple as possible.

--
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: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Thu, 12 Jun 2008 05:05:33 GMT
View Forum Message <> Reply to Message

On Wed, 11 Jun 2008 01:48:20 -0700
"Paul Menage" <menage@google.com> wrote:

> On Wed, Jun 11, 2008 at 1:27 AM, KAMEZAWA Hiroyuki
> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > Sorry. try another sentense..
> >
> > I think cgroup itself is designed to be able to be used without middleware.

>
> True, but it shouldn't be hostile to middleware, since I think that
> automated use will be much more common. (And certainly if you count
> the number of servers :-))
>
> > IOW, whether using middleware or not is the matter of users not of developpers.
> > There will be a system that system admin controlles all and move tasks by hand.
> > ex)...personal notebooks etc..
> >
>
> You think so? I think that at the very least users will be using tools
> based around config scripts, rule engines and libcgroup, if not a
> persistent daemon.
>
I believe some users will never use middlewares because of their special
usage of linux.

> >> If the common mode for middleware starting a new cgroup is fork() /
> >> move / exec() then after the fork(), the child will be sharing pages
> >> with the main daemon process. So the move will pull all the daemon's
> >> memory into the new cgroup
> >>
> > My patch (this patch) just moves Private Anon page to new cgroup. (of mapcount=1)
>
> OK, well that makes it more reasonable regarding the above problem.
> But I can still see problems if, say, a single thread moves into a new
> cgroup, you move the entire memory. Perhaps you should only do so if
> the mm->owner changes task?
>

Thank you for pointing out. I'll add mm->owner check.

BTW, should we have a cgroup for SYSVIPC resource controller and devide it
from memory resource controller ? I think that per-task on-demand usage
accounting is not suitable for shmem (and hugepage).
per-creater (caller of shmget()) accounting seems to be better for me.

Just a question:
What happens when a thread (not thread-group-leader) changes its ns by
ns-cgroup ? not-allowed ?

Thanks,
-Kame

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by yamamoto on Thu, 12 Jun 2008 05:20:33 GMT
View Forum Message <> Reply to Message

> > i think that you can redirect new charges in TASK to DEST
> > so that usage_of_task(TASK) will not grow.
> >
>
> Hmm, to do that, we have to handle complicated cgroup's attach ops.
>
> at this moving, memcg is pointed by
> - TASK->cgroup->memcg(CURR)
> after move
> - TASK->another_cgroup->memcg(DEST)
>
> This move happens before cgroup is replaced by another_cgroup.

currently cgroup_attach_task calls ->attach callbacks after
assigning tsk->cgroups. are you talking about something else?

YAMAMOTO Takashi

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

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Thu, 12 Jun 2008 06:50:35 GMT
View Forum Message <> Reply to Message

On Thu, 12 Jun 2008 14:20:33 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > > i think that you can redirect new charges in TASK to DEST
> > > so that usage_of_task(TASK) will not grow.
> > >
> >
> > Hmm, to do that, we have to handle complicated cgroup's attach ops.
> >

> > at this moving, memcg is pointed by
> > - TASK->cgroup->memcg(CURR)
> > after move
> > - TASK->another_cgroup->memcg(DEST)
> >
> > This move happens before cgroup is replaced by another_cgroup.
>
> currently cgroup_attach_task calls ->attach callbacks after
> assigning tsk->cgroups. are you talking about something else?
>

Sorry, I move all in can_attach(). s/attach/can_attach

Thanks,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by serue on Thu, 12 Jun 2008 13:17:48 GMT
View Forum Message <> Reply to Message

Quoting KAMEZAWA Hiroyuki (kamezawa.hiroyu@jp.fujitsu.com):
> On Wed, 11 Jun 2008 01:48:20 -0700
> "Paul Menage" <menage@google.com> wrote:
>
> > On Wed, Jun 11, 2008 at 1:27 AM, KAMEZAWA Hiroyuki
> > <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > > Sorry. try another sentense..
> > >
> > > I think cgroup itself is designed to be able to be used without middleware.
> >
> > True, but it shouldn't be hostile to middleware, since I think that
> > automated use will be much more common. (And certainly if you count
> > the number of servers :-))
> >
> > > IOW, whether using middleware or not is the matter of users not of developpers.
> > > There will be a system that system admin controlles all and move tasks by hand.
> > > ex)...personal notebooks etc..
> > >
> >
> > You think so? I think that at the very least users will be using tools
> > based around config scripts, rule engines and libcgroup, if not a
> > persistent daemon.

> >
> I believe some users will never use middlewares because of their special
> usage of linux.
>
>
>
> > >> If the common mode for middleware starting a new cgroup is fork() /
> > >> move / exec() then after the fork(), the child will be sharing pages
> > >> with the main daemon process. So the move will pull all the daemon's
> > >> memory into the new cgroup
> > >>
> > > My patch (this patch) just moves Private Anon page to new cgroup. (of mapcount=1)
> >
> > OK, well that makes it more reasonable regarding the above problem.
> > But I can still see problems if, say, a single thread moves into a new
> > cgroup, you move the entire memory. Perhaps you should only do so if
> > the mm->owner changes task?
> >
>
> Thank you for pointing out. I'll add mm->owner check.
>
> BTW, should we have a cgroup for SYSVIPC resource controller and devide it
> from memory resource controller ? I think that per-task on-demand usage
> accounting is not suitable for shmem (and hugepage).
> per-creater (caller of shmget()) accounting seems to be better for me.
>
> Just a question:
> What happens when a thread (not thread-group-leader) changes its ns by
> ns-cgroup ? not-allowed ?

I don't quite understand the question. I assume you're asking whether
your cgroup, when composed with ns, will refuse a task in cgroup /cg/1/2
from being able to

 mkdir /cg/1/2/3
 echo $$ > /cg/1/2/3/tasks

or

 unshare(CLONE_NEWNS)

which the ns cgroup would allow, and what your cgroup would do in that
case. If your question ("not-allowed ?") is about ns cgroup behavior
then please rephrase.

thanks,
-serge

_____

Subject: Re: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Thu, 12 Jun 2008 13:34:48 GMT
View Forum Message <> Reply to Message

----- Original Message -----
>> Just a question:
>> What happens when a thread (not thread-group-leader) changes its ns by
>> ns-cgroup ? not-allowed ?
>
>I don't quite understand the question. I assume you're asking whether
>your cgroup, when composed with ns, will refuse a task in cgroup /cg/1/2
>from being able to
>
> mkdir /cg/1/2/3
> echo $$ > /cg/1/2/3/tasks
>
>or
>
> unshare(CLONE_NEWNS)
>
>which the ns cgroup would allow, and what your cgroup would do in that
>case. If your question ("not-allowed ?") is about ns cgroup behavior
>then please rephrase.

Ah, sorry. I'm just curious. (and I should read the code before making
quiestion.)

Assume a thread group contains threadA, threadB, threadC.

I wanted to ask "Can threadA, and threadB, and threadC
be in different cgroups ? And if so, how ns cgroup handles it ?"

Maybe I don't understand ns cgroup.

Thanks,
-Kame

_____
Containers mailing list

Subject: Re: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by serue on Thu, 12 Jun 2008 21:08:12 GMT
View Forum Message <> Reply to Message

Quoting kamezawa.hiroyu@jp.fujitsu.com (kamezawa.hiroyu@jp.fujitsu.com):
> ----- Original Message -----
> >> Just a question:
> >> What happens when a thread (not thread-group-leader) changes its ns by
> >> ns-cgroup ? not-allowed ?
> >
> >I don't quite understand the question. I assume you're asking whether
> >your cgroup, when composed with ns, will refuse a task in cgroup /cg/1/2
> >from being able to
> >
> > mkdir /cg/1/2/3
> > echo $$ > /cg/1/2/3/tasks
> >
> >or
> >
> > unshare(CLONE_NEWNS)
> >
> >which the ns cgroup would allow, and what your cgroup would do in that
> >case. If your question ("not-allowed ?") is about ns cgroup behavior
> >then please rephrase.
>
> Ah, sorry. I'm just curious. (and I should read the code before making
> quiestion.)
>
> Assume a thread group contains threadA, threadB, threadC.
>
> I wanted to ask "Can threadA, and threadB, and threadC
> be in different cgroups ? And if so, how ns cgroup handles it ?"
>
> Maybe I don't understand ns cgroup.

In part yes, but nonetheless a very interesting question when it comes
to composition of cgroups!

Yes, you can have threads in different cgroups. The ns cgroup just
tracks nsproxy unshares. So if you run the attached program and look
around, you'll see the first thread is in /cg/taskpid while the second
one is in /cg/taskpid/secondthreadpid.

Clearly, composing this with a cgroup which needs to keep threads in the

same cgroup becomes problematic!

Interesting :)

-serge

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
#include <libgen.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

#include <linux/unistd.h>

#ifndef SYS_unshare
#ifdef __NR_unshare
#define SYS_unshare __NR_unshare
#elif __i386__
#define SYS_unshare 310
#elif __ia64__
#define SYS_unshare 1296
#elif __x86_64__
#define SYS_unshare 272
#elif __s390x__
#define SYS_unshare 303
#elif __powerpc__
#define SYS_unshare 282
#else
#error "unshare not supported on this architecure."
#endif
#endif

#define CSIGNAL         0x000000ff      /* signal mask to be sent at exit */
#define CLONE_VM        0x00000100      /* set if VM shared between processes */
#define CLONE_FS        0x00000200      /* set if fs info shared between processes */
#define CLONE_FILES     0x00000400      /* set if open files shared between processes */
#define CLONE_SIGHAND   0x00000800      /* set if signal handlers and blocked signals shared
*/
#define CLONE_PTRACE    0x00002000      /* set if we want to let tracing continue on the child
too */
#define CLONE_VFORK     0x00004000      /* set if the parent wants the child to wake it up on
```

```
mm_release
*/
#define CLONE_PARENT    0x00008000      /* set if we want to have the same parent as the
cloner */
#define CLONE_THREAD    0x00010000      /* Same thread group? */
#define CLONE_NEWNS     0x00020000      /* New namespace group? */
#define CLONE_SYSVSEM   0x00040000      /* share system V SEM_UNDO semantics */
#define CLONE_SETTLS    0x00080000      /* create a new TLS for the child */
#define CLONE_PARENT_SETTID     0x00100000      /* set the TID in the parent */
#define CLONE_CHILD_CLEARTID    0x00200000      /* clear the TID in the child */
#define CLONE_DETACHED          0x00400000      /* Unused, ignored */
#define CLONE_UNTRACED          0x00800000      /* set if the tracing process can't force
CLONE_PTRACE on
 this clone */
#define CLONE_CHILD_SETTID      0x01000000      /* set the TID in the child */
#define CLONE_STOPPED           0x02000000      /* Start in stopped state */
#define CLONE_NEWUTS            0x04000000      /* New utsname group? */
#define CLONE_NEWIPC            0x08000000      /* New ipcs */
#define CLONE_NEWNUSER          0x10000000      /* New level 2 network namespace */
#define CLONE_NEWPID            0x20000000      /* New pid namespace */

int child2(void *data)
{
 sleep(500);
}

int child1(void *data)
{
 int stacksize = 8*getpagesize();
 void *childstack, *stack = malloc(stacksize);
 unsigned long flags;
 int ret;

 if (!stack) {
  perror("malloc");
  return -1;
 }
 childstack = stack + stacksize;

 flags = CLONE_THREAD | CLONE_VM | CLONE_SIGHAND | CLONE_NEWNS |
CLONE_NEWUTS;
 ret = clone(child2, childstack, flags, NULL);
 if (ret == -1) {
  perror("clone2");
  return -1;
 }

 sleep(500);
```

```
}

int main(int argc, char *argv[])
{
 int stacksize = 4*getpagesize();
 int pid, ret, status;
 void *childstack, *stack = malloc(stacksize);
 unsigned long flags;

 if (!stack) {
  perror("malloc");
  return -1;
 }
 childstack = stack + stacksize;

 flags = CLONE_NEWNS | CLONE_NEWUTS;
 ret = clone(child1, childstack, flags, (void *)argv);
 if (ret == -1) {
  perror("clone");
  return -1;
 }

 pid = ret;
 while ((ret = waitpid(pid, &status, __WALL) != -1)) {
  printf("pid %d, status %d, ret %d\n",
    pid, status, ret);
 };
 printf("pid %d exited with status %d\n", pid, status);
 exit(0);
}
```

_____

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Fri, 13 Jun 2008 00:31:12 GMT
View Forum Message <> Reply to Message

On Thu, 12 Jun 2008 16:08:12 -0500
"Serge E. Hallyn" <serue@us.ibm.com> wrote:

> > Assume a thread group contains threadA, threadB, threadC.
> >
> > I wanted to ask "Can threadA, and threadB, and threadC
> > be in different cgroups ? And if so, how ns cgroup handles it ?"
> >

> > Maybe I don't understand ns cgroup.
>
> In part yes, but nonetheless a very interesting question when it comes
> to composition of cgroups!
>
> Yes, you can have threads in different cgroups. The ns cgroup just
> tracks nsproxy unshares. So if you run the attached program and look
> around, you'll see the first thread is in /cg/taskpid while the second
> one is in /cg/taskpid/secondthreadpid.
>
> Clearly, composing this with a cgroup which needs to keep threads in the
> same cgroup becomes problematic!
>
> Interesting :)
>

Thank you for kindly explanation. I'll take this into account. I confirmed
memory resouce controller should not get tasks's cgroup directly from "task"
and should get it from "mm->owner".

Thank you.

Regards,
-Kame

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

---

Subject: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by KAMEZAWA Hiroyuki on Fri, 13 Jun 2008 00:37:43 GMT
View Forum Message <> Reply to Message

On Fri, 13 Jun 2008 09:34:36 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> Thank you for kindly explanation. I'll take this into account. I confirmed
> memory resouce controller should not get tasks's cgroup directly from "task"
> and should get it from "mm->owner".
>
And this means the whole thread group's memory related cgroup can be changed
when mm->owner is changed. I'm not sure this is not a problem but it seems
complex.

Thanks,
-Kame

_____

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