Subject: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Balbir Singh on Fri, 05 Oct 2007 04:14:06 GMT View Forum Message <> Reply to Message

Found-by: Hugh Dickins <hugh@veritas.com>

mem\_cgroup\_charge() in unuse\_pte() is called under a lock, the pte\_lock. That's clearly incorrect, since we pass GFP\_KERNEL to mem\_cgroup\_charge() for allocation of page\_cgroup.

This patch release the lock and reacquires the lock after the call to mem\_cgroup\_charge().

Tested on a powerpc box by calling swapoff in the middle of a cgroup running a workload that pushes pages to swap.

```
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
```

```
diff -puN mm/swapfile.c~memory-controller-fix-unuse-pte-charging mm/swapfile.c
--- linux-2.6.23-rc8/mm/swapfile.c~memory-controller-fix-unuse-pte-charging 2007-10-03
13:45:56.000000000 +0530
```

+++ linux-2.6.23-rc8-balbir/mm/swapfile.c 2007-10-05 08:49:54.000000000 +0530

@ @ -507,11 +507,18 @ @ unsigned int count\_swap\_pages(int type,

\* just let do\_wp\_page work it out if a write is requested later - to

```
* force COW, vm_page_prot omits write permission from any private vma.
*/
```

```
-static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
```

```
- unsigned long addr, swp_entry_t entry, struct page *page)
```

```
+static int unuse_pte(struct vm_area_struct *vma, pte_t *pte, pmd_t *pmd,
```

```
+ unsigned long addr, swp_entry_t entry, struct page *page,
```

```
+ spinlock_t **ptl)
```

```
{
```

```
- if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
```

```
+ pte_unmap_unlock(pte - 1, *ptl);
```

+

```
+ if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL)) {
```

```
+ pte_offset_map_lock(vma->vm_mm, pmd, addr, ptl);
return -ENOMEM;
```

```
+ }
```

```
+
```

+ pte\_offset\_map\_lock(vma->vm\_mm, pmd, addr, ptl);

```
inc_mm_counter(vma->vm_mm, anon_rss);
```

```
get_page(page);
@ @ -543,7 +550,8 @ @ static int unuse pte range(struct vm are
  * Test inline before going to call unuse_pte.
  */
 if (unlikely(pte_same(*pte, swp_pte))) {
- ret = unuse_pte(vma, pte++, addr, entry, page);
+ ret = unuse pte(vma, pte++, pmd, addr, entry, page,
    &ptl);
+
  break;
 }
 } while (pte++, addr += PAGE_SIZE, addr != end);
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL
Containers mailing list
```

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

Subject: Re: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Hugh Dickins on Sun, 07 Oct 2007 16:57:32 GMT View Forum Message <> Reply to Message

On Fri, 5 Oct 2007, Balbir Singh wrote:

>

> Found-by: Hugh Dickins <hugh@veritas.com>

>

> mem\_cgroup\_charge() in unuse\_pte() is called under a lock, the pte\_lock. That's

> clearly incorrect, since we pass GFP\_KERNEL to mem\_cgroup\_charge() for

> allocation of page\_cgroup.

>

> This patch release the lock and reacquires the lock after the call to

> mem\_cgroup\_charge().

>

> Tested on a powerpc box by calling swapoff in the middle of a cgroup

> running a workload that pushes pages to swap.

Hard to test it adequately at present, while that call to mem\_cgroup\_charge is never allocating anything new.

Sorry, it's a bit ugly (the intertwining of unuse\_pte and its caller), it's got a bug, and fixing that bug makes it uglier.

The bug is that you ignore the pte ptr returned by pte\_offset\_map\_lock: we could be preempted on to a different cpu just there, so a different cpu's kmap\_atomic area used, with a different pte pointer; which would need to be passed back to the caller for when it unmaps.

I much prefer my patch appended further down: considering how it's safe for you to drop the ptl there because of holding page lock, pushed me into seeing that we can actually do our scanning without ptl, which in many configurations has the advantage of staying preemptible (though preemptible swapoff is not terribly high on anyone's ticklist ;).

But you may well prefer that we split it into two: with me taking responsibility and blame for the preliminary patch which relaxes the locking, and you then adding the mem\_cgroup\_charge (and the exceptional mem\_cgroup\_uncharge\_page) on top of that.

Hugh

>

```
> Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
> ---
> 
> mm/swapfile.c | 16 +++++++++----
```

```
> 1 file changed, 12 insertions(+), 4 deletions(-)
```

>

> diff -puN mm/swapfile.c~memory-controller-fix-unuse-pte-charging mm/swapfile.c > --- linux-2.6.23-rc8/mm/swapfile.c~memory-controller-fix-unuse-pte-charging 2007-10-03 13:45:56.000000000 +0530

```
> +++ linux-2.6.23-rc8-balbir/mm/swapfile.c 2007-10-05 08:49:54.000000000 +0530
```

- > @ @ -507,11 +507,18 @ @ unsigned int count\_swap\_pages(int type,
- > \* just let do\_wp\_page work it out if a write is requested later to
- > \* force COW, vm\_page\_prot omits write permission from any private vma.

> \*/

```
> -static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
```

> - unsigned long addr, swp\_entry\_t entry, struct page \*page)

```
> +static int unuse_pte(struct vm_area_struct *vma, pte_t *pte, pmd_t *pmd,
```

```
> + unsigned long addr, swp_entry_t entry, struct page *page,
```

```
> + spinlock_t **ptl)
> {
```

```
> - if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
```

```
> + pte_unmap_unlock(pte - 1, *ptl);
```

> +

```
> + if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL)) {
```

```
> + pte_offset_map_lock(vma->vm_mm, pmd, addr, ptl);
```

```
> return -ENOMEM;
```

- > + }
- > +

```
> + pte offset map lock(vma->vm mm, pmd, addr, ptl);
>
> inc_mm_counter(vma->vm_mm, anon_rss);
> get_page(page);
> @ @ -543,7 +550,8 @ @ static int unuse_pte_range(struct vm_are
    * Test inline before going to call unuse_pte.
>
    */
>
>
  if (unlikely(pte_same(*pte, swp_pte))) {
> ret = unuse pte(vma, pte++, addr, entry, page);
> + ret = unuse pte(vma, pte++, pmd, addr, entry, page,
> +
      &ptl);
    break:
>
   }
>
  } while (pte++, addr += PAGE_SIZE, addr != end);
>
--- 2.6.23-rc8-mm2/mm/swapfile.c 2007-09-27 12:03:36.000000000 +0100
+++ linux/mm/swapfile.c 2007-10-07 14:33:05.000000000 +0100
@ @ -507,11 +507,23 @ @ unsigned int count_swap_pages(int type,
 * just let do wp page work it out if a write is requested later - to
 * force COW, vm page prot omits write permission from any private vma.
 */
-static int unuse pte(struct vm area struct *vma, pte t *pte,
+static int unuse_pte(struct vm_area_struct *vma, pmd_t *pmd,
 unsigned long addr, swp_entry_t entry, struct page *page)
{
+ spinlock_t *ptl;
+ pte_t *pte;
+ int ret = 1;
+
 if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
- return -ENOMEM:
+ ret = -ENOMEM;
+
+ pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
+ if (ret > 0)
+ mem_cgroup_uncharge_page(page);
+ ret = 0:
+ goto out;
+ }
 inc_mm_counter(vma->vm_mm, anon_rss);
 get_page(page);
@ @ -524,7 +536,9 @ @ static int unuse_pte(struct vm_area_stru
 * immediately swapped out again after swapon.
 */
 activate_page(page);
- return 1;
```

```
+out:
+ pte_unmap_unlock(pte, ptl);
+ return ret;
}
static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@ @ -533,21 +547,33 @ @ static int unuse_pte_range(struct vm_are
{
 pte_t swp_pte = swp_entry_to_pte(entry);
 pte t *pte;
- spinlock_t *ptl;
 int ret = 0:
- pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ /*
+ * We don't actually need pte lock while scanning for swp_pte:
+ * since we hold page lock, swp pte cannot be inserted into or
+ * removed from a page table while we're scanning; but on some
+ * architectures (e.g. i386 with PAE) we might catch a glimpse
+ * of unmatched parts which look like swp pte, so unuse pte
+ * must recheck under pte lock. Scanning without the lock
+ * is preemptible if CONFIG PREEMPT without CONFIG HIGHPTE.
+ */
+ pte = pte_offset_map(pmd, addr);
 do {
 /*
  * swapoff spends a _lot_ of time in this loop!
  * Test inline before going to call unuse_pte.
  */
 if (unlikely(pte_same(*pte, swp_pte))) {
- ret = unuse pte(vma, pte++, addr, entry, page);
- break:
+ pte_unmap(pte);
+ ret = unuse_pte(vma, pmd, addr, entry, page);
+ if (ret)
+ goto out;
+ pte = pte_offset_map(pmd, addr);
 }
 } while (pte++, addr += PAGE_SIZE, addr != end);
- pte_unmap_unlock(pte - 1, ptl);
+ pte_unmap(pte - 1);
+out:
 return ret:
}
```

Containers mailing list Containers@lists.linux-foundation.org Subject: Re: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse pte() Posted by Balbir Singh on Sun, 07 Oct 2007 17:48:31 GMT View Forum Message <> Reply to Message Hugh Dickins wrote: > On Fri, 5 Oct 2007, Balbir Singh wrote: >> Found-by: Hugh Dickins <hugh@veritas.com> >> >> mem\_cgroup\_charge() in unuse\_pte() is called under a lock, the pte\_lock. That's >> clearly incorrect, since we pass GFP\_KERNEL to mem\_cgroup\_charge() for >> allocation of page cgroup. >> >> This patch release the lock and reacquires the lock after the call to >> mem\_cgroup\_charge(). >> >> Tested on a powerpc box by calling swapoff in the middle of a cgroup >> running a workload that pushes pages to swap. > > Hard to test it adequately at present, while that call > to mem\_cgroup\_charge is never allocating anything new. > Yes, your right! > Sorry, it's a bit ugly (the intertwining of unuse\_pte and its caller), > it's got a bug, and fixing that bug makes it uglier. > > The bug is that you ignore the pte ptr returned by pte offset map lock: > we could be preempted on to a different cpu just there, so a different > cpu's kmap atomic area used, with a different pte pointer; which would > need to be passed back to the caller for when it unmaps. > Good point! I forgot that we unmap the pte when we unlock it > I much prefer my patch appended further down: considering how it's safe > for you to drop the ptl there because of holding page lock, pushed me > into seeing that we can actually do our scanning without ptl, which in > many configurations has the advantage of staying preemptible (though

> preemptible swapoff is not terribly high on anyone's ticklist ;).

>

I like your patch, my comments on it below

- > But you may well prefer that we split it into two: with me taking
- > responsibility and blame for the preliminary patch which relaxes
- > the locking, and you then adding the mem\_cgroup\_charge (and the
- > exceptional mem\_cgroup\_uncharge\_page) on top of that.

>

Sounds good, you could submit both parts to Andrew. I think Andrew would like to split up the patches as well, so that the major change of scanning without the lock and the memory controller fix are two different patches. My changes are pretty trivial and well covered under your patch.

```
> Hugh
>
>> Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
>> ----
>>
>> mm/swapfile.c | 16 +++++++++++++----
>> 1 file changed, 12 insertions(+), 4 deletions(-)
>>
>> diff -puN mm/swapfile.c~memory-controller-fix-unuse-pte-charging mm/swapfile.c
>> --- linux-2.6.23-rc8/mm/swapfile.c~memory-controller-fix-unuse-pte-charging 2007-10-03
13:45:56.00000000 +0530
>> +++ linux-2.6.23-rc8-balbir/mm/swapfile.c 2007-10-05 08:49:54.000000000 +0530
>> @ @ -507,11 +507,18 @ @ unsigned int count_swap_pages(int type,
>> * just let do wp page work it out if a write is requested later - to
>> * force COW, vm_page_prot omits write permission from any private vma.
>> */
>> -static int unuse pte(struct vm area struct *vma, pte t *pte,
>> - unsigned long addr, swp_entry_t entry, struct page *page)
>> +static int unuse_pte(struct vm_area_struct *vma, pte_t *pte, pmd_t *pmd,
>> + unsigned long addr, swp_entry_t entry, struct page *page,
>> + spinlock_t **ptl)
>> {
>> - if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
>> + pte_unmap_unlock(pte - 1, *ptl);
>> +
>> + if (mem cgroup charge(page, vma->vm mm, GFP KERNEL)) {
>> + pte_offset_map_lock(vma->vm_mm, pmd, addr, ptl);
>> return -ENOMEM;
>> + }
>> +
>> + pte_offset_map_lock(vma->vm_mm, pmd, addr, ptl);
>>
>> inc_mm_counter(vma->vm_mm, anon_rss);
>> get_page(page);
>> @ @ -543,7 +550,8 @ @ static int unuse pte range(struct vm are
     * Test inline before going to call unuse pte.
>>
```

```
*/
>>
>> if (unlikely(pte_same(*pte, swp_pte))) {
>> - ret = unuse_pte(vma, pte++, addr, entry, page);
>> + ret = unuse_pte(vma, pte++, pmd, addr, entry, page,
       &ptl);
>> +
     break;
>>
    }
>>
   } while (pte++, addr += PAGE_SIZE, addr != end);
>>
>
> --- 2.6.23-rc8-mm2/mm/swapfile.c 2007-09-27 12:03:36.000000000 +0100
> +++ linux/mm/swapfile.c 2007-10-07 14:33:05.000000000 +0100
> @ @ -507,11 +507,23 @ @ unsigned int count swap pages(int type,
 * just let do_wp_page work it out if a write is requested later - to
>
  * force COW, vm_page_prot omits write permission from any private vma.
>
> */
> -static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
> +static int unuse pte(struct vm area struct *vma, pmd t *pmd,
   unsigned long addr, swp_entry_t entry, struct page *page)
>
> {
> + spinlock_t *ptl;
> + pte_t *pte;
> + int ret = 1;
> +
> if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
> - return -ENOMEM;
> + ret = -ENOMEM;
> +
```

With this change I think, ret = mem\_cgroup\_charge(....) makes more sense

> + pte = pte\_offset\_map\_lock(vma->vm\_mm, pmd, addr, &ptl);

We map the pte in unuse\_pte\_range() and unmap it before calling this routing, can't we keep it mapped and acquire the lock here? Looking through pte\_offset\_map\_lock, it did not seem like mapping and acquiring the lock were indivisible operations.

```
> + if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
> + if (ret > 0)
> + mem_cgroup_uncharge_page(page);
```

Then we can check for ret  $\geq 0$ 

> + ret = 0; > + goto out;

```
> + }
>
> inc_mm_counter(vma->vm_mm, anon_rss);
> get_page(page);
> @ @ -524,7 +536,9 @ @ static int unuse_pte(struct vm_area_stru
   * immediately swapped out again after swapon.
>
   */
>
> activate_page(page);
> - return 1;
> +out:
> + pte_unmap_unlock(pte, ptl);
> + return ret:
> }
>
> static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
> @ @ -533,21 +547,33 @ @ static int unuse_pte_range(struct vm_are
> {
> pte_t swp_pte = swp_entry_to_pte(entry);
> pte t *pte;
> - spinlock t *ptl;
> int ret = 0;
>
> - pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
> + /*
> + * We don't actually need pte lock while scanning for swp_pte:
> + * since we hold page lock, swp_pte cannot be inserted into or
> + * removed from a page table while we're scanning; but on some
> + * architectures (e.g. i386 with PAE) we might catch a glimpse
> + * of unmatched parts which look like swp pte, so unuse pte
> + * must recheck under pte lock. Scanning without the lock
> + * is preemptible if CONFIG PREEMPT without CONFIG HIGHPTE.
> + */
> + pte = pte_offset_map(pmd, addr);
 do {
>
   /*
>
    * swapoff spends a lot of time in this loop!
>
    * Test inline before going to call unuse_pte.
>
    */
>
   if (unlikely(pte_same(*pte, swp_pte))) {
>
> ret = unuse_pte(vma, pte++, addr, entry, page);
> - break;
> + pte_unmap(pte);
> + ret = unuse_pte(vma, pmd, addr, entry, page);
> + if (ret)
> + goto out;
> + pte = pte_offset_map(pmd, addr);
> }
> } while (pte++, addr += PAGE SIZE, addr != end);
```

> - pte\_unmap\_unlock(pte - 1, ptl);
> + pte\_unmap(pte - 1);
> +out:
> return ret;
> }
>

---

>

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: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Balbir Singh on Mon, 15 Oct 2007 17:27:14 GMT View Forum Message <> Reply to Message

Hugh Dickins wrote:

```
> --- 2.6.23-rc8-mm2/mm/swapfile.c 2007-09-27 12:03:36.000000000 +0100
> +++ linux/mm/swapfile.c 2007-10-07 14:33:05.000000000 +0100
> @ @ -507,11 +507,23 @ @ unsigned int count_swap_pages(int type,
> * just let do wp page work it out if a write is requested later - to
  * force COW, vm page prot omits write permission from any private vma.
>
  */
>
> -static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
> +static int unuse pte(struct vm area struct *vma, pmd t *pmd,
  unsigned long addr, swp entry t entry, struct page *page)
>
> {
> + spinlock_t *ptl;
> + pte_t * pte;
> + int ret = 1;
> +
> if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
> - return -ENOMEM:
> + ret = -ENOMEM;
> +
> + pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
> + if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
> +  if (ret > 0)
> + mem_cgroup_uncharge_page(page);
```

```
> + ret = 0;
> + goto out;
> + }
>
> inc_mm_counter(vma->vm_mm, anon_rss);
> get_page(page);
> @ @ -524.7 +536.9 @ @ static int unuse pte(struct vm area stru
  * immediately swapped out again after swapon.
>
  */
>
 activate page(page);
>
> - return 1;
> +out:
> + pte_unmap_unlock(pte, ptl);
> + return ret;
> }
>
> static int unuse pte range(struct vm area struct *vma, pmd t *pmd,
> @ @ -533,21 +547,33 @ @ static int unuse_pte_range(struct vm_are
> {
> pte_t swp_pte = swp_entry_to_pte(entry);
> pte_t *pte;
> - spinlock t *ptl;
> int ret = 0;
>
> - pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
> + /*
> + * We don't actually need pte lock while scanning for swp_pte:
> + * since we hold page lock, swp pte cannot be inserted into or
> + * removed from a page table while we're scanning; but on some
> + * architectures (e.g. i386 with PAE) we might catch a glimpse
> + * of unmatched parts which look like swp pte, so unuse pte
> + * must recheck under pte lock. Scanning without the lock
> + * is preemptible if CONFIG_PREEMPT without CONFIG_HIGHPTE.
> + */
> + pte = pte_offset_map(pmd, addr);
> do {
>
  /*
    * swapoff spends a lot of time in this loop!
>
    * Test inline before going to call unuse_pte.
>
    */
>
   if (unlikely(pte same(*pte, swp pte))) {
>
> ret = unuse_pte(vma, pte++, addr, entry, page);
> - break;
> + pte_unmap(pte);
> + ret = unuse_pte(vma, pmd, addr, entry, page);
> + if (ret)
> + goto out;
> + pte = pte offset map(pmd, addr);
```

```
> }
> }
> }
while (pte++, addr += PAGE_SIZE, addr != end);
> - pte_unmap_unlock(pte - 1, ptl);
> + pte_unmap(pte - 1);
> +out:
> return ret;
> }
>
```

I tested this patch and it seems to be working fine. I tried swapoff -a in the middle of tests consuming swap. Not 100% rigorous, but a good test nevertheless.

Tested-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: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Hugh Dickins on Mon, 22 Oct 2007 18:51:33 GMT View Forum Message <> Reply to Message

On Mon, 15 Oct 2007, Balbir Singh wrote:

> Hugh Dickins wrote:

>>

>> --- 2.6.23-rc8-mm2/mm/swapfile.c 2007-09-27 12:03:36.000000000 +0100

>>+++ linux/mm/swapfile.c 2007-10-07 14:33:05.000000000 +0100

>> @ @ -507,11 +507,23 @ @ unsigned int count\_swap\_pages(int type,

>> \* just let do\_wp\_page work it out if a write is requested later - to

>> \* force COW, vm\_page\_prot omits write permission from any private vma.

>> \*/

>> -static int unuse\_pte(struct vm\_area\_struct \*vma, pte\_t \*pte,

> > +static int unuse\_pte(struct vm\_area\_struct \*vma, pmd\_t \*pmd,

>> unsigned long addr, swp\_entry\_t entry, struct page \*page)

···· >

- > I tested this patch and it seems to be working fine. I tried swapoff -a
- > in the middle of tests consuming swap. Not 100% rigorous, but a good

> test nevertheless.

>

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

Thanks, Balbir. Sorry for the delay. I've not forgotten our agreement that I should be splitting it into before-and-after mem cgroup patches. But it's low priority for me until we're genuinely assigning to a cgroup there. Hope to get back to looking into that tomorrow, but no promises.

I think you still see no problem, where I claim that simply omitting the mem charge mods from mm/swap\_state.c leads to OOMs? Maybe our difference is because my memhog in the cgroup is using more memory than RAM, not just more memory than allowed to the cgroup. I suspect that arrives at a state (when the swapcache pages are not charged) where it cannot locate the pages it needs to reclaim to stay within its limit.

Hugh

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

Subject: Re: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Balbir Singh on Wed, 24 Oct 2007 12:14:42 GMT View Forum Message <> Reply to Message

Hugh Dickins wrote: > On Mon, 15 Oct 2007, Balbir Singh wrote: >> Hugh Dickins wrote: >>> --- 2.6.23-rc8-mm2/mm/swapfile.c 2007-09-27 12:03:36.000000000 +0100 >>> +++ linux/mm/swapfile.c 2007-10-07 14:33:05.000000000 +0100 >>> @ @ -507,11 +507,23 @ @ unsigned int count swap pages(int type, >>> \* just let do\_wp\_page work it out if a write is requested later - to >>> \* force COW, vm\_page\_prot omits write permission from any private vma. >>> \*/ >>> -static int unuse\_pte(struct vm\_area\_struct \*vma, pte\_t \*pte, >>> +static int unuse\_pte(struct vm\_area\_struct \*vma, pmd\_t \*pmd, unsigned long addr, swp entry t entry, struct page \*page) >>> > .... >> I tested this patch and it seems to be working fine. I tried swapoff -a >> in the middle of tests consuming swap. Not 100% rigorous, but a good >> test nevertheless. >> >> Tested-by: Balbir Singh <balbir@linux.vnet.ibm.com>

>

- > Thanks, Balbir. Sorry for the delay. I've not forgotten our
   > agreement that I should be splitting it into before-and-after
   > mem cgroup patches. But it's low priority for me until we're
   > genuinely assigning to a cgroup there. Hope to get back to
- > looking into that tomorrow, but no promises.

>

No Problem. We have some time with this one.

I think you still see no problem, where I claim that simply
omitting the mem charge mods from mm/swap\_state.c leads to OOMs?
Maybe our difference is because my memhog in the cgroup is using
more memory than RAM, not just more memory than allowed to the
cgroup. I suspect that arrives at a state (when the swapcache
pages are not charged) where it cannot locate the pages it needs
to reclaim to stay within its limit.

Yes, in my case there I use memory less than RAM and more than that is allowed by the cgroup. It's quite possible that in your case the swapcache has grown significantly without any limit/control on it. The memhog program is using memory at a rate much higher than the rate of reclaim. Could you share your memhog program, please? In the use case you've mentioned/tested, having these mods to control swapcache is actually useful, right?

Could you share your major objections at this point with the memory controller at this point. I hope to be able to look into/resolve them as my first priority in my list of items to work on.

> Hugh

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: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte()

On Wed, 24 Oct 2007, Balbir Singh wrote:

> Hugh Dickins wrote:

> >

> > Thanks, Balbir. Sorry for the delay. I've not forgotten our

- > > agreement that I should be splitting it into before-and-after
- > > mem cgroup patches. But it's low priority for me until we're
- > > genuinely assigning to a cgroup there. Hope to get back to
- > > looking into that tomorrow, but no promises.

>

> No Problem. We have some time with this one.

Phew - I still haven't got there.

> I think you still see no problem, where I claim that simply
> omitting the mem charge mods from mm/swap\_state.c leads to OOMs?
> Maybe our difference is because my memhog in the cgroup is using
> more memory than RAM, not just more memory than allowed to the
> cgroup. I suspect that arrives at a state (when the swapcache
> pages are not charged) where it cannot locate the pages it needs
> to reclaim to stay within its limit.

Yes, in my case there I use memory less than RAM and more than that
 is allowed by the cgroup. It's quite possible that in your case the
 swapcache has grown significantly without any limit/control on it.

- > The membog program is using memory at a rate much higher than the
- > rate of reclaim. Could you share your memhog program, please?

Gosh, it's nothing special. Appended below, but please don't shame me by taking it too seriously. Defaults to working on a 600M mmap because I'm in the habit of booting mem=512M. You probably have something better yourself that you'd rather use.

> In the use case you've mentioned/tested, having these mods to

> control swapcache is actually useful, right?

No idea what you mean by "these mods to control swapcache"?

With your mem\_cgroup mods in mm/swap\_state.c, swapoff assigns the pages read in from swap to whoever's running swapoff and your unuse\_pte mem\_cgroup\_charge never does anything useful: swap pages should get assigned to the appropriate cgroups at that point.

Without your mem\_cgroup mods in mm/swap\_state.c, unuse\_pte makes the right assignments (I believe). But I find that swapout (using 600M in a 512M machine) from a 200M cgroup quickly OOMs, whereas it behaves correctly with your mm/swap\_state.c. Thought little yet about what happens to shmem swapped pages, and swap readahead pages; but still suspect that they and the above issue will need a "limbo" cgroup, for pages which are expected to belong to a not-yet-identified mem cgroup.

>

- > Could you share your major objections at this point with the memory
- > controller at this point. I hope to be able to look into/resolve them
- > as my first priority in my list of items to work on.

The things I've noticed so far, as mentioned before and above.

But it does worry me that I only came here through finding swapoff broken by that unuse\_mm return value, and then found one issue after another. It feels like the mem cgroup people haven't really thought through or tested swap at all, and that if I looked further I'd uncover more.

That's simply FUD, and I apologize if I'm being unfair: but that is how it feels, and I expect we all know that phase in a project when solving one problem uncovers three - suggests it's not ready.

```
Hugh
```

```
/* swapout.c */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <svs/mman.h>
int main(int argc, char *argv[])
{
unsigned long *base = (unsigned long *)0x08400000;
unsigned long size;
unsigned long limit;
unsigned long i;
char *ptr = NULL;
size = argv[1]? strtoul(argv[1], &ptr, 0): 600;
if (size >= 3*1024)
 size = 0;
size *= 1024*1024;
limit = size / sizeof(unsigned long);
if (size == 0 \parallel base + limit + 1024 > \&size) {
 errno = EINVAL;
 perror("swapout");
```

```
exit(1);
}
base = mmap(base, size, PROT_READ|PROT_WRITE,
    MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);
if (base == (unsigned long *)(-1)) {
 perror("mmap");
 exit(1);
}
for (i = 0; i < \text{limit}; i++)
 base[i] = i;
if (ptr && *ptr == '.') {
 printf("Type <Return> to continue ");
 fflush(stdout);
 getchar();
}
for (i = 0; i < limit; i++)
 base[i] = limit - i;
return 0;
}
```

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

Subject: Re: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Balbir Singh on Fri, 26 Oct 2007 06:14:44 GMT View Forum Message <> Reply to Message

Hugh Dickins wrote:

```
> Gosh, it's nothing special. Appended below, but please don't shame
```

> me by taking it too seriously. Defaults to working on a 600M mmap

- > because I'm in the habit of booting mem=512M. You probably have
- > something better yourself that you'd rather use.

>

Thanks for sending it. I do have something more generic that I got from my colleague.

>> In the use case you've mentioned/tested, having these mods to >> control swapcache is actually useful, right?

>

> No idea what you mean by "these mods to control swapcache"?

>

Yes

> With your mem\_cgroup mods in mm/swap\_state.c, swapoff assigns

> the pages read in from swap to whoever's running swapoff and your

> unuse\_pte mem\_cgroup\_charge never does anything useful: swap pages

> should get assigned to the appropriate cgroups at that point.

>

> Without your mem\_cgroup mods in mm/swap\_state.c, unuse\_pte makes

> the right assignments (I believe). But I find that swapout (using

> 600M in a 512M machine) from a 200M cgroup quickly OOMs, whereas

> it behaves correctly with your mm/swap\_state.c.

>

I'll try this test and play with your test

> Thought little yet about what happens to shmem swapped pages,

> and swap readahead pages; but still suspect that they and the

> above issue will need a "limbo" cgroup, for pages which are

> expected to belong to a not-yet-identified mem cgroup.

>

This is something I am yet to experiment with. I suspect this should be easy to do if we decide to go this route.

>> Could you share your major objections at this point with the memory>> controller at this point. I hope to be able to look into/resolve them>> as my first priority in my list of items to work on.

>

> The things I've noticed so far, as mentioned before and above.

> But it does worry me that I only came here through finding swapoff
> broken by that unuse\_mm return value, and then found one issue
> after another. It feels like the mem cgroup people haven't really
> thought through or tested swap at all, and that if I looked further

> I'd uncover more.

>

I thought so far that you've found a couple of bugs and one issue with the way we account for swapcache. Other users, KAMEZAWA, YAMAMOTO have been using and enhancing the memory controller. I can point you to a set of links where I posted all the test results. Swap was tested mostly through swapout/swapin when the cgroup goes over limit. Please do help uncover as many bugs as possible, please look more closely as you find more time.

> That's simply FUD, and I apologize if I'm being unfair: but that

> is how it feels, and I expect we all know that phase in a project

> when solving one problem uncovers three - suggests it's not ready.

I disagree, all projects/code do have bugs, which we are trying to resolve, but I don't think there are any major design drawbacks that \*cannot\* be fixed. We discussed the design at VM-Summit and everyone agreed it was the way to go forward (even though Double LRU has its complexity).

> Hugh

[snip]

Thanks for the review and your valuable feedback!

--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: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Balbir Singh on Sun, 28 Oct 2007 20:32:19 GMT View Forum Message <> Reply to Message

On Mon, Oct 29, 2007 at 01:57:40AM +0530, Balbir Singh wrote: Hugh Dickins wrote:

[snip]

> Without your mem\_cgroup mods in mm/swap\_state.c, unuse\_pte makes

> the right assignments (I believe). But I find that swapout (using

> 600M in a 512M machine) from a 200M cgroup quickly OOMs, whereas

> it behaves correctly with your mm/swap\_state.c.

>

On my UML setup, I booted the UML instance with 512M of memory and used the swapout program that you shared. I tried two things

- 1. Ran swapout without any changes. The program ran well without any OOM condition occuring, lot of reclaim occured.
- 2. Ran swapout with the changes to mm/swap\_state.c removed (diff below)

and I still did not see any OOM. The reclaim count was much lesser since swap cache did not get accounted back to the cgroup from which pages were being evicted.

I am not sure why I don't see the OOM that you see, still trying. May be I missing something obvious at this late hour in the night :-)

Output of the tests

-----

balbir@ubuntu:/container/swapout\$ cat memory.limit in bytes 209715200 balbir@ubuntu:/container/swapout\$ cat memory.usage\_in\_bytes 65536 balbir@ubuntu:/container/swapout\$ cat tasks 1815 1847 balbir@ubuntu:/container/swapout\$ ps PID TTY TIME CMD 1815 pts/0 00:00:00 bash 1848 pts/0 00:00:00 ps balbir@ubuntu:/container/swapout\$ ~/swapout balbir@ubuntu:/container/swapout\$ echo \$? 0 balbir@ubuntu:/container/swapout\$ cat memory.failcnt 18 Diff to remove mods from swap state.c (for testing only) --- mm/swap state.c.org 2007-10-29 01:42:14.000000000 +0530 +++ mm/swap state.c 2007-10-29 01:52:48.000000000 +0530 @ @ -79.10 +79.6 @ @ static int add to swap cache(struct pa BUG\_ON(PageSwapCache(page)); BUG\_ON(PagePrivate(page)); - error = mem\_cgroup\_cache\_charge(page, current->mm, gfp\_mask); - if (error) - goto out; error = radix tree preload(gfp mask); if (!error) { write\_lock\_irg(&swapper\_space.tree\_lock); @ @ -94,14 +90,11 @ @ static int \_\_add\_to\_swap\_cache(struct pa set\_page\_private(page, entry.val); total\_swapcache\_pages++; inc zone page state(page, NR FILE PAGES); - } else

```
    mem_cgroup_uncharge_page(page);

+ }
 write_unlock_irg(&swapper_space.tree_lock);
 radix_tree_preload_end();
- } else

    mem_cgroup_uncharge_page(page);

-out:
+ }
 return error;
}
@ @ -141,7 +134,6 @ @ void __delete_from_swap_cache(struct pag
 BUG_ON(PageWriteback(page));
 BUG_ON(PagePrivate(page));
- mem cgroup uncharge page(page);
 radix_tree_delete(&swapper_space.page_tree, page_private(page));
 set page private(page, 0);
 ClearPageSwapCache(page);
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: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Hugh Dickins on Mon, 29 Oct 2007 21:07:34 GMT View Forum Message <> Reply to Message

On Mon, 29 Oct 2007, Balbir Singh wrote:

> On Mon, Oct 29, 2007 at 01:57:40AM +0530, Balbir Singh wrote:

> Hugh Dickins wrote:

```
>
```

> [snip]

>

- > > Without your mem\_cgroup mods in mm/swap\_state.c, unuse\_pte makes
- > > the right assignments (I believe). But I find that swapout (using
- > > 600M in a 512M machine) from a 200M cgroup quickly OOMs, whereas
- > > it behaves correctly with your mm/swap\_state.c.
- > >
- >

> On my UML setup, I booted the UML instance with 512M of memory and

> used the swapout program that you shared. I tried two things

> >

> 1. Ran swapout without any changes. The program ran well without

- > any OOM condition occuring, lot of reclaim occured.
- > 2. Ran swapout with the changes to mm/swap\_state.c removed (diff below)
- > and I still did not see any OOM. The reclaim count was much lesser
- > since swap cache did not get accounted back to the cgroup from
- > which pages were being evicted.
- >
- > I am not sure why I don't see the OOM that you see, still trying. May be
- > I missing something obvious at this late hour in the night :-)

I reconfirm that I do see those OOMs. I'll have to try harder to analyze how they come about: I sure don't expect you to debug a problem you cannot reproduce. But what happens if you try it native rather than using UML?

Hugh

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

Subject: Re: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Balbir Singh on Mon, 29 Oct 2007 22:01:38 GMT View Forum Message <> Reply to Message

Hugh Dickins wrote: > On Mon, 29 Oct 2007, Balbir Singh wrote: >> On Mon, Oct 29, 2007 at 01:57:40AM +0530, Balbir Singh wrote: >> Hugh Dickins wrote: >> >> [snip] >> >>> Without your mem\_cgroup mods in mm/swap\_state.c, unuse\_pte makes >>> the right assignments (I believe). But I find that swapout (using >>> 600M in a 512M machine) from a 200M cgroup guickly OOMs, whereas >>> it behaves correctly with your mm/swap state.c. >>> >> On my UML setup, I booted the UML instance with 512M of memory and >> used the swapout program that you shared. I tried two things >> >> >> 1. Ran swapout without any changes. The program ran well without

>> any OOM condition occuring, lot of reclaim occured.

>> 2. Ran swapout with the changes to mm/swap\_state.c removed (diff below)

>> and I still did not see any OOM. The reclaim count was much lesser

>> since swap cache did not get accounted back to the cgroup from

>> which pages were being evicted.

>>

>> I am not sure why I don't see the OOM that you see, still trying. May be >> I missing something obvious at this late hour in the night :-)

>

> I reconfirm that I do see those OOMs. I'll have to try harder to

> analyze how they come about: I sure don't expect you to debug a

> problem you cannot reproduce. But what happens if you try it

> native rather than using UML?

>

> Hugh

On a real box - a powerpc machine that I have access to

- 1. I don't see the OOM with the mods removed (I have swap space at-least twice of RAM with mem=512M, I have at-least 1G of swap).
- 2. Running under the container is much much faster than running swapout in the root container. The machine is almost unusable if swapout is run under the root container

At this momemnt, I suspect one of two things

- 1. Our mods to swap\_state.c are different
- 2. Our configuration is different, main-memory to swap-size ratio

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: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte() Posted by Hugh Dickins on Tue, 30 Oct 2007 16:57:58 GMT View Forum Message <> Reply to Message

On Tue, 30 Oct 2007, Balbir Singh wrote:

- >
- > At this momemnt, I suspect one of two things
- >
- > 1. Our mods to swap\_state.c are different

I believe they're the same (just take swap\_state.c back to how it was without mem\_cgroup mods) - or would be, if after finding this effect I hadn't added a "swap\_in\_cg" switch to move between the two behaviours to study it better (though I do need to remember to swapoff and swapon between the two: sometimes I do forget).

> 2. Our configuration is different, main-memory to swap-size ratio

I doubt the swapsize is relevant: just so long as there's some (a little more than 200M I guess); I've got 1GB-2GB on different boxes.

There may well be something about our configs that's significantly different. I'd failed to mention SMP (4 cpu), and that I happen to have /proc/sys/vm/swappiness 100; but find it happens on UP also, and when I go back to default swappiness 60.

I've reordered your mail for more dramatic effect...

>

> On a real box - a powerpc machine that I have access to

I've tried on 3 Intel and 1 PowerPC now: the Intels show the OOMs and the PowerPC does not. I rather doubt it's an Intel versus PowerPC issue as such, but interesting that we see the same.

## >

- > 1. I don't see the OOM with the mods removed (I have swap
- > space at-least twice of RAM with mem=512M, I have at-least
- > 1G of swap).

mem=512M with 1G of swap, yes, I'm the same.

- > 2. Running under the container is much much faster than running
- > swapout in the root container. The machine is almost unusable
- > if swapout is run under the root container

That's rather interesting, isn't it? Probably irrelevant to the OOM issue we're investigating, but worthy of investigation in itself.

Maybe I saw the same on the PowerPC: I simply forgot to set up the cgroup one time, and my sequence of three swapouts (sometimes only two out of three OOM, on those boxes that do OOM) seemed to take a very long time (but I wasn't trying to do anything else on it at the same time, so didn't notice if it was "unusable").

I'll probe on.

Hugh

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

Subject: Re: [RFC] [-mm PATCH] Memory controller fix swap charging context in unuse\_pte()

Posted by Balbir Singh on Tue, 30 Oct 2007 18:28:20 GMT View Forum Message <> Reply to Message

Hugh Dickins wrote:

> On Tue, 30 Oct 2007, Balbir Singh wrote:

>> At this momemnt, I suspect one of two things

>>

>> 1. Our mods to swap\_state.c are different

>

> I believe they're the same (just take swap\_state.c back to how it

> was without mem\_cgroup mods) - or would be, if after finding this

> effect I hadn't added a "swap\_in\_cg" switch to move between the

> two behaviours to study it better (though I do need to remember

> to swapoff and swapon between the two: sometimes I do forget).

>

>> 2. Our configuration is different, main-memory to swap-size ratio

> I doubt the swapsize is relevant: just so long as there's some (a

> little more than 200M I guess); I've got 1GB-2GB on different boxes.

>

I agree, just wanted to make sure that there is enough swap

There may well be something about our configs that's significantly
 different. I'd failed to mention SMP (4 cpu), and that I happen
 to have /proc/sys/vm/swappiness 100; but find it happens on UP
 also, and when I go back to default swappiness 60.

OK.. so those are out of the equation

> I've reordered your mail for more dramatic effect...

>> On a real box - a powerpc machine that I have access to >

> I've tried on 3 Intel and 1 PowerPC now: the Intels show the OOMs> and the PowerPC does not. I rather doubt it's an Intel versus

> PowerPC issue as such, but interesting that we see the same.

Very surprising, I am surprised that it's architecture dependent. Let me try and grab an Intel box and try.

>> 1. I don't see the OOM with the mods removed (I have swap space at-least twice of RAM - with mem=512M, I have at-least >> 1G of swap). >> > > mem=512M with 1G of swap, yes, I'm the same. > >> 2. Running under the container is much much faster than running swapout in the root container. The machine is almost unusable >> if swapout is run under the root container >> > > That's rather interesting, isn't it? Probably irrelevant to the > OOM issue we're investigating, but worthy of investigation in itself. >

Yes, it irrelevant, but I find it to be a good use case for using the memory controller :-) I found that kswapd running at prio -5, seemed to hog quite a bit of the CPU. But it needs more independent investigation, like you've suggested.

> Maybe I saw the same on the PowerPC: I simply forgot to set up the
> cgroup one time, and my sequence of three swapouts (sometimes only
> two out of three OOM, on those boxes that do OOM) seemed to take a
> very long time (but I wasn't trying to do anything else on it at
> the same time, so didn't notice if it was "unusable").

> I'll probe on.

>

Me too.. I'll try and acquire a good x86\_64 box and test on it.

> Hugh

>

> --

> To unsubscribe, send a message with 'unsubscribe linux-mm' in

- > the body to majordomo@kvack.org. For more info on Linux MM,
- > see: http://www.linux-mm.org/ .
- > Don't email: <a href=mailto:"dont@kvack.org"> email@kvack.org </a>

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

Page 27 of 27 ---- Generated from OpenVZ Forum