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
13:45:56.000000000 +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;
> +

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 >= 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