Subject: [PATCH 1/6 mm] swapoff: scan ptes preemptibly
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:08:48 GMT
View Forum Message <> Reply to Message

Provided that CONFIG_HIGHPTE is not set, unuse_pte_range can reduce latency
in swapoff by scanning the page table preemptibly: so long as unuse_pte is
careful to recheck that entry under pte lock.

(To tell the truth, this patch was not inspired by any cries for lower
latency here: rather, this restructuring permits a future memory controller
patch to allocate with GFP_KERNEL in unuse_pte, where before it could not.
But it would be wrong to tuck this change away inside a memcgroup patch.)

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

This patch could go anywhere in the mm series before the memory-controller
patches: I suggest just after swapin-fix-valid-swaphandles-defect.patch
Subsequent patches N/6 go in different places amongst the memory-controller
patches: please see accompanying suggestions.

 mm/swapfile.c | 38 ++++++++++++++++++++++++++++++++------
 1 file changed, 31 insertions(+), 7 deletions(-)

```
--- patch0/mm/swapfile.c 2007-11-07 19:41:45.000000000 +0000
+++ patch1/mm/swapfile.c 2007-11-08 12:34:12.000000000 +0000
@@ -506,9 +506,19 @@ 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 void 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 found = 1;
+
+ pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
+  found = 0;
+  goto out;
+ }
+
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
@@ -520,6 +530,9 @@ static void unuse_pte(struct vm_area_str
   * immediately swapped out again after swapon.
```

```
   */
  activate_page(page);
+out:
+ pte_unmap_unlock(pte, ptl);
+ return found;
 }

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -528,22 +541,33 @@ static int unuse_pte_range(struct vm_are
 {
  pte_t swp_pte = swp_entry_to_pte(entry);
  pte_t *pte;
- spinlock_t *ptl;
  int found = 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 and mmap_sem, swp_pte cannot be inserted into the
+  * page table while we're scanning; though it could get zapped, and on
+  * some architectures (e.g. x86_32 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 pte lock lets it be
+  * preemptible whenever CONFIG_PREEMPT but not 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))) {
-   unuse_pte(vma, pte++, addr, entry, page);
-   found = 1;
-   break;
+   pte_unmap(pte);
+   found = unuse_pte(vma, pmd, addr, entry, page);
+   if (found)
+    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 found;
 }
```

Subject: [PATCH 2/6 mm] memcgroup: temporarily revert swapoff mod
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:10:23 GMT
View Forum Message <> Reply to Message

Whaaa? This patch precisely reverts the "swapoff: scan ptes preemptibly"
patch just presented. It's a temporary measure to allow existing memory
controller patches to apply without rejects: in due course they should be
rendered down into one sensible patch, and this reversion disappear.

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

This patch should go immediately before the memory-controller patches,
or immediately before memory-controller-memory-accounting-v7.patch

mm/swapfile.c | 38 +++++++----------------------------
1 file changed, 7 insertions(+), 31 deletions(-)

```
--- patch1/mm/swapfile.c 2007-11-08 12:34:12.000000000 +0000
+++ patch2/mm/swapfile.c 2007-11-08 12:34:12.000000000 +0000
@@ -506,19 +506,9 @@ 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, pmd_t *pmd,
+static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
   unsigned long addr, swp_entry_t entry, struct page *page)
 {
- spinlock_t *ptl;
- pte_t *pte;
- int found = 1;
-
- pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
- if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
-  found = 0;
-  goto out;
- }
-
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
@@ -530,9 +520,6 @@ static int unuse_pte(struct vm_area_stru
   * immediately swapped out again after swapon.
```

```
   */
  activate_page(page);
-out:
- pte_unmap_unlock(pte, ptl);
- return found;
 }

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -541,33 +528,22 @@ static int unuse_pte_range(struct vm_are
 {
  pte_t swp_pte = swp_entry_to_pte(entry);
  pte_t *pte;
+ spinlock_t *ptl;
  int found = 0;

- /*
-  * We don't actually need pte lock while scanning for swp_pte: since
-  * we hold page lock and mmap_sem, swp_pte cannot be inserted into the
-  * page table while we're scanning; though it could get zapped, and on
-  * some architectures (e.g. x86_32 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 pte lock lets it be
-  * preemptible whenever CONFIG_PREEMPT but not CONFIG_HIGHPTE.
-  */
- pte = pte_offset_map(pmd, addr);
+ pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
  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))) {
-   pte_unmap(pte);
-   found = unuse_pte(vma, pmd, addr, entry, page);
-   if (found)
-    goto out;
-   pte = pte_offset_map(pmd, addr);
+   unuse_pte(vma, pte++, addr, entry, page);
+   found = 1;
+   break;
   }
  } while (pte++, addr += PAGE_SIZE, addr != end);
- pte_unmap(pte - 1);
-out:
+ pte_unmap_unlock(pte - 1, ptl);
  return found;
 }
```

Subject: [PATCH 3/6 mm] memcgroup: fix try_to_free order
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:11:12 GMT
View Forum Message <> Reply to Message

Why does try_to_free_mem_cgroup_pages try for order 1 pages? It's called
when mem_cgroup_charge_common would go over the limit, and that's adding
an order 0 page. I see no reason: it has to be a typo: fix it.

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

Insert just after memory-controller-add-per-container-lru-and-reclaim-v7.patch

mm/vmscan.c | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

```
--- patch2/mm/vmscan.c 2007-11-08 15:46:21.000000000 +0000
+++ patch3/mm/vmscan.c 2007-11-08 15:48:08.000000000 +0000
@@ -1354,7 +1354,7 @@ unsigned long try_to_free_mem_cgroup_pag
   .may_swap = 1,
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
-  .order = 1,
+  .order = 0,
   .mem_cgroup = mem_cont,
   .isolate_pages = mem_cgroup_isolate_pages,
  };
```


Subject: [PATCH 4/6 mm] memcgroup: reinstate swapoff mod
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:12:03 GMT
View Forum Message <> Reply to Message

This patch reinstates the "swapoff: scan ptes preemptibly" mod we started
with: in due course it should be rendered down into the earlier patches,
leaving us with a more straightforward mem_cgroup_charge mod to unuse_pte,
allocating with GFP_KERNEL while holding no spinlock and no atomic kmap.

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

Insert just after memory-controller-make-charging-gfp-mask-aware.patch or
you may prefer to insert 4-6 all together before memory-cgroup-enhancements

```
 mm/swapfile.c |   42 ++++++++++++++++++++++++++++++++++--------
 1 file changed, 34 insertions(+), 8 deletions(-)

--- patch3/mm/swapfile.c 2007-11-08 15:48:08.000000000 +0000
+++ patch4/mm/swapfile.c 2007-11-08 15:55:12.000000000 +0000
@@ -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 and mmap_sem, swp_pte cannot be inserted into the
+  * page table while we're scanning; though it could get zapped, and on
+  * some architectures (e.g. x86_32 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 pte lock lets it be
+  * preemptible whenever CONFIG_PREEMPT but not 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;
 }
```


Subject: [PATCH 5/6 mm] memcgroup: fix zone isolation OOM
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:13:22 GMT
View Forum Message <> Reply to Message

mem_cgroup_charge_common shows a tendency to OOM without good reason,
when a memhog goes well beyond its rss limit but with plenty of swap
available. Seen on x86 but not on PowerPC; seen when the next patch
omits swapcache from memcgroup, but we presume it can happen without.

mem_cgroup_isolate_pages is not quite satisfying reclaim's criteria
for OOM avoidance. Already it has to scan beyond the nr_to_scan limit
when it finds a !LRU page or an active page when handling inactive or
an inactive page when handling active. It needs to do exactly the same
when it finds a page from the wrong zone (the x86 tests had two zones,
the PowerPC tests had only one).

Don't increment scan and then decrement it in these cases, just move
the incrementation down. Fix recent off-by-one when checking against
nr_to_scan. Cut out "Check if the meta page went away from under us",
presumably left over from early debugging: no amount of such checks
could save us if this list really were being updated without locking.

This change does make the unlimited scan while holding two spinlocks
even worse - bad for latency and bad for containment; but that's a
separate issue which is better left to be fixed a little later.

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

Insert just after
bugfix-for-memory-cgroup-controller-avoid-pagelru-page-in-mem_cgroup_isolate_pages-fix.patch
or just before memory-cgroup-enhancements

 mm/memcontrol.c | 17 ++++++-------------
 1 file changed, 4 insertions(+), 13 deletions(-)

```
--- patch4/mm/memcontrol.c 2007-11-08 16:03:33.000000000 +0000
+++ patch5/mm/memcontrol.c 2007-11-08 16:51:39.000000000 +0000
@@ -260,24 +260,20 @@ unsigned long mem_cgroup_isolate_pages(u
  spin_lock(&mem_cont->lru_lock);
  scan = 0;
  list_for_each_entry_safe_reverse(pc, tmp, src, lru) {
-  if (scan++ > nr_to_scan)
+  if (scan >= nr_to_scan)
    break;
   page = pc->page;
   VM_BUG_ON(!pc);

-  if (unlikely(!PageLRU(page))) {
-   scan--;
+  if (unlikely(!PageLRU(page)))
    continue;
-  }
```

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

@@ -288,13 +284,8 @@ unsigned long mem_cgroup_isolate_pages(u
   if (page_zone(page) != z)
    continue;

-  /*
-   * Check if the meta page went away from under us
-   */
-  if (!list_empty(&pc->lru))
-   list_move(&pc->lru, &pc_list);
-  else
-   continue;
+  scan++;
+  list_move(&pc->lru, &pc_list);

   if (__isolate_lru_page(page, mode) == 0) {
    list_move(&page->lru, dst);
```

Subject: [PATCH 6/6 mm] memcgroup: revert swap_state mods
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:14:22 GMT
View Forum Message <> Reply to Message

If we're charging rss and we're charging cache, it seems obvious that we should be charging swapcache - as has been done. But in practice that doesn't work out so well: both swapin readahead and swapoff leave the majority of pages charged to the wrong cgroup (the cgroup that happened to read them in, rather than the cgroup to which they belong).

(Which is why unuse_pte's GFP_KERNEL while holding pte lock never showed up as a problem: no allocation was ever done there, every page read being already charged to the cgroup which initiated the swapoff.)

It all works rather better if we leave the charging to do_swap_page and
unuse_pte, and do nothing for swapcache itself: revert mm/swap_state.c
to what it was before the memory-controller patches. This also speeds
up significantly a contained process working at its limit: because it
no longer needs to keep waiting for swap writeback to complete.

Is it unfair that swap pages become uncharged once they're unmapped,
even though they're still clearly private to particular cgroups? For
a short while, yes; but PageReclaim arranges for those pages to go to
the end of the inactive list and be reclaimed soon if necessary.

shmem/tmpfs pages are a distinct case: their charging also benefits
from this change, but their second life on the lists as swapcache
pages may prove more unfair - that I need to check next.

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

Insert just after 5/6: the tree builds okay if it goes earlier, just after
memory-controller-bug_on.patch, but 5/6 fixes OOM made more likely by 6/6.
Alternatively, hand edit all of the mm/swap_state.c mods out of all of the
memory-controller patches which modify it.

 mm/swap_state.c | 15 ++----------
 1 file changed, 2 insertions(+), 13 deletions(-)

--- patch5/mm/swap_state.c 2007-11-08 15:58:50.000000000 +0000
+++ patch6/mm/swap_state.c 2007-11-08 16:01:11.000000000 +0000
@@ -17,7 +17,6 @@
 #include <linux/backing-dev.h>
 #include <linux/pagevec.h>
 #include <linux/migrate.h>
-#include <linux/memcontrol.h>

 #include <asm/pgtable.h>

@@ -79,11 +78,6 @@ static int __add_to_swap_cache(struct pa
  BUG_ON(!PageLocked(page));
  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_irq(&swapper_space.tree_lock);
@@ -95,14 +89,10 @@ 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_irq(&swapper_space.tree_lock);
   radix_tree_preload_end();
- } else
-  mem_cgroup_uncharge_page(page);
-out:
+ }
  return error;
 }

@@ -143,7 +133,6 @@ 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);
```

Subject: Re: [PATCH 6/6 mm] memcgroup: revert swap_state mods
Posted by KAMEZAWA Hiroyuki on Fri, 09 Nov 2007 09:21:56 GMT
View Forum Message <> Reply to Message

On Fri, 9 Nov 2007 07:14:22 +0000 (GMT)
Hugh Dickins <hugh@veritas.com> wrote:

> If we're charging rss and we're charging cache, it seems obvious that
> we should be charging swapcache - as has been done. But in practice
> that doesn't work out so well: both swapin readahead and swapoff leave
> the majority of pages charged to the wrong cgroup (the cgroup that
> happened to read them in, rather than the cgroup to which they belong).
>

Thank you. I welcome this patch :)

Could I confirm a change in the logic ?

* Before this patch, wrong swapcache charge is added to one who
  called try_to_free_page().

* After this patch, anonymous page's charge will drop to 0 when
  page_remove_rmap() is called.

Regards,
-Kame

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

---

Subject: Re: [PATCH 5/6 mm] memcgroup: fix zone isolation OOM
Posted by KAMEZAWA Hiroyuki on Fri, 09 Nov 2007 09:27:29 GMT
View Forum Message <> Reply to Message

On Fri, 9 Nov 2007 07:13:22 +0000 (GMT)
Hugh Dickins <hugh@veritas.com> wrote:

> mem_cgroup_charge_common shows a tendency to OOM without good reason,
> when a memhog goes well beyond its rss limit but with plenty of swap
> available. Seen on x86 but not on PowerPC; seen when the next patch
> omits swapcache from memcgroup, but we presume it can happen without.
>
> mem_cgroup_isolate_pages is not quite satisfying reclaim's criteria
> for OOM avoidance. Already it has to scan beyond the nr_to_scan limit
> when it finds a !LRU page or an active page when handling inactive or
> an inactive page when handling active. It needs to do exactly the same
> when it finds a page from the wrong zone (the x86 tests had two zones,
> the PowerPC tests had only one).
>
> Don't increment scan and then decrement it in these cases, just move
> the incrementation down. Fix recent off-by-one when checking against
> nr_to_scan. Cut out "Check if the meta page went away from under us",
> presumably left over from early debugging: no amount of such checks
> could save us if this list really were being updated without locking.
>
> This change does make the unlimited scan while holding two spinlocks
> even worse - bad for latency and bad for containment; but that's a
> separate issue which is better left to be fixed a little later.
>

Okay, I agree with this logic for scan.

I'll consider some kind of optimization for avoiding all list scan
because of a zone's page is not included in cgroup's lru.

Maybe counting the number of active/inactive per zone (or per node) will
be first help.

Thanks,
-Kame

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

---

Subject: Re: [PATCH 6/6 mm] memcgroup: revert swap_state mods
Posted by Hugh Dickins on Mon, 12 Nov 2007 04:57:03 GMT
View Forum Message <> Reply to Message

On Fri, 9 Nov 2007, KAMEZAWA Hiroyuki wrote:
> On Fri, 9 Nov 2007 07:14:22 +0000 (GMT)
> Hugh Dickins <hugh@veritas.com> wrote:
>
> > If we're charging rss and we're charging cache, it seems obvious that
> > we should be charging swapcache - as has been done. But in practice
> > that doesn't work out so well: both swapin readahead and swapoff leave
> > the majority of pages charged to the wrong cgroup (the cgroup that
> > happened to read them in, rather than the cgroup to which they belong).
>
> Thank you. I welcome this patch :)

Thank you! But perhaps less welcome if I don't confirm...

> Could I confirm a change in the logic ?
>
> * Before this patch, wrong swapcache charge is added to one who
> called try_to_free_page().

try_to_free_pages? No, I don't think any wrong charge was made
there. It was when reading in swap pages. The usual way is by
swapin_readahead, which reads in a cluster of swap pages, which
are quite likely to belong to different memcgroups, but were all
charged to the one which is doing the fault on its target page.
Another way is in swapoff, where they all got charged to whoever
was doing the swapoff (and the charging in unuse_pte was a no-op).

> * After this patch, anonymous page's charge will drop to 0 when
> page_remove_rmap() is called.

Yes, when its final (usually its only) page_remove_rmap is called.

Hugh

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

---

Subject: Re: [PATCH 1/6 mm] swapoff: scan ptes preemptibly
Posted by Balbir Singh on Mon, 12 Nov 2007 05:04:02 GMT
View Forum Message <> Reply to Message

Hugh Dickins wrote:
> Provided that CONFIG_HIGHPTE is not set, unuse_pte_range can reduce latency
> in swapoff by scanning the page table preemptibly: so long as unuse_pte is
> careful to recheck that entry under pte lock.
>
> (To tell the truth, this patch was not inspired by any cries for lower
> latency here: rather, this restructuring permits a future memory controller
> patch to allocate with GFP_KERNEL in unuse_pte, where before it could not.
> But it would be wrong to tuck this change away inside a memcgroup patch.)
>
> Signed-off-by: Hugh Dickins <hugh@veritas.com>
> ---

Looks good to me

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com> and earlier
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: [PATCH 3/6 mm] memcgroup: fix try_to_free order
Posted by Balbir Singh on Mon, 12 Nov 2007 05:05:34 GMT
View Forum Message <> Reply to Message

Hugh Dickins wrote:
> Why does try_to_free_mem_cgroup_pages try for order 1 pages? It's called
> when mem_cgroup_charge_common would go over the limit, and that's adding
> an order 0 page. I see no reason: it has to be a typo: fix it.
>
> Signed-off-by: Hugh Dickins <hugh@veritas.com>
> ---
> Insert just after memory-controller-add-per-container-lru-and-reclaim-v7.patch
>
> mm/vmscan.c | 2 +-
> 1 file changed, 1 insertion(+), 1 deletion(-)
>
> --- patch2/mm/vmscan.c 2007-11-08 15:46:21.000000000 +0000
> +++ patch3/mm/vmscan.c 2007-11-08 15:48:08.000000000 +0000
> @@ -1354,7 +1354,7 @@ unsigned long try_to_free_mem_cgroup_pag
> .may_swap = 1,
> .swap_cluster_max = SWAP_CLUSTER_MAX,
> .swappiness = vm_swappiness,
> - .order = 1,
> + .order = 0,
> .mem_cgroup = mem_cont,
> .isolate_pages = mem_cgroup_isolate_pages,
> };

Thanks for catching this, it is a typo

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

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

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

---

Subject: Re: [PATCH 4/6 mm] memcgroup: reinstate swapoff mod
Posted by Balbir Singh on Mon, 12 Nov 2007 05:08:50 GMT
View Forum Message <> Reply to Message

Hugh Dickins wrote:
> This patch reinstates the "swapoff: scan ptes preemptibly" mod we started
> with: in due course it should be rendered down into the earlier patches,
> leaving us with a more straightforward mem_cgroup_charge mod to unuse_pte,
> allocating with GFP_KERNEL while holding no spinlock and no atomic kmap.

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

Looks good to me

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

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

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

---

Subject: Re: [PATCH 6/6 mm] memcgroup: revert swap_state mods
Posted by KAMEZAWA Hiroyuki on Mon, 12 Nov 2007 05:17:54 GMT
View Forum Message <> Reply to Message

On Mon, 12 Nov 2007 04:57:03 +0000 (GMT)
Hugh Dickins <hugh@veritas.com> wrote:
> > Could I confirm a change in the logic ?
> >
> > * Before this patch, wrong swapcache charge is added to one who
> > called try_to_free_page().
>
> try_to_free_pages? No, I don't think any wrong charge was made
> there.
Ah, sorry. I misundestood when it was added to swapcache...

> It was when reading in swap pages. The usual way is by
> swapin_readahead, which reads in a cluster of swap pages, which
> are quite likely to belong to different memcgroups, but were all
> charged to the one which is doing the fault on its target page.
> Another way is in swapoff, where they all got charged to whoever
> was doing the swapoff (and the charging in unuse_pte was a no-op).
>
I see.

> > * After this patch, anonymous page's charge will drop to 0 when
> > page_remove_rmap() is called.
>
> Yes, when its final (usually its only) page_remove_rmap is called.
>
Thank you for confirmation !

Regards,
-Kame

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

---

Subject: Re: [PATCH 5/6 mm] memcgroup: fix zone isolation OOM
Posted by Balbir Singh on Mon, 12 Nov 2007 06:42:57 GMT
View Forum Message <> Reply to Message

Hugh Dickins wrote:
> mem_cgroup_charge_common shows a tendency to OOM without good reason,
> when a memhog goes well beyond its rss limit but with plenty of swap
> available. Seen on x86 but not on PowerPC; seen when the next patch
> omits swapcache from memcgroup, but we presume it can happen without.
>
> mem_cgroup_isolate_pages is not quite satisfying reclaim's criteria
> for OOM avoidance. Already it has to scan beyond the nr_to_scan limit
> when it finds a !LRU page or an active page when handling inactive or
> an inactive page when handling active. It needs to do exactly the same
> when it finds a page from the wrong zone (the x86 tests had two zones,
> the PowerPC tests had only one).
>
> Don't increment scan and then decrement it in these cases, just move
> the incrementation down. Fix recent off-by-one when checking against
> nr_to_scan. Cut out "Check if the meta page went away from under us",
> presumably left over from early debugging: no amount of such checks
> could save us if this list really were being updated without locking.
>

It's a spill over from the old code, we do all operations under
the mem_cont's lru_lock.

> This change does make the unlimited scan while holding two spinlocks
> even worse - bad for latency and bad for containment; but that's a
> separate issue which is better left to be fixed a little later.
>
> Signed-off-by: Hugh Dickins <hugh@veritas.com>

For the swapout test case scenario sent by Hugh

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: [PATCH 6/6 mm] memcgroup: revert swap_state mods
Posted by Balbir Singh on Mon, 12 Nov 2007 06:56:04 GMT
View Forum Message <> Reply to Message

Hugh Dickins wrote:
> If we're charging rss and we're charging cache, it seems obvious that
> we should be charging swapcache - as has been done. But in practice
> that doesn't work out so well: both swapin readahead and swapoff leave
> the majority of pages charged to the wrong cgroup (the cgroup that
> happened to read them in, rather than the cgroup to which they belong).
>
> (Which is why unuse_pte's GFP_KERNEL while holding pte lock never
> showed up as a problem: no allocation was ever done there, every page
> read being already charged to the cgroup which initiated the swapoff.)
>
> It all works rather better if we leave the charging to do_swap_page and
> unuse_pte, and do nothing for swapcache itself: revert mm/swap_state.c
> to what it was before the memory-controller patches. This also speeds
> up significantly a contained process working at its limit: because it
> no longer needs to keep waiting for swap writeback to complete.
>

Yes, it does speed up things, but we lose control over swap cache.
It might grow very large, but having said that I am in favour of
removing the mods till someone faces a severe problem with them.
Another approach is to provide a per-container tunable as to
whether swap cache should be controlled or not and document
the side-effects of swap cache control.

> Is it unfair that swap pages become uncharged once they're unmapped,
> even though they're still clearly private to particular cgroups? For
> a short while, yes; but PageReclaim arranges for those pages to go to
> the end of the inactive list and be reclaimed soon if necessary.
>
> shmem/tmpfs pages are a distinct case: their charging also benefits
> from this change, but their second life on the lists as swapcache
> pages may prove more unfair - that I need to check next.
>

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

Thanks for the patch

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

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

_____

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