Subject: Re: [RFC][PATCH 1/2] split up shrink_page_list()
Posted by serue on Thu, 14 Jun 2007 16:00:09 GMT

View Forum Message <> Reply to Message

Quoting Dave Hansen (hansendc@us.ibm.com):
>
> This patch takes shrink_page_list() and splits out its main
> loop into another function: try_to_shrink_page(). I think
> this makes it all a bit more readable.

Haven't checked every error path yet, but it looks correct enough
to these mm-virgin eyes.

Why is try_to_put_page_in_swap() defined in this patch? Is that
on purpose?

thanks,
-serge

```
> ---
>
>  lxc-dave/mm/vmscan.c |  276 +++++++++++++++++++++++++++------------------------
>  1 file changed, 150 insertions(+), 126 deletions(-)
>
> diff -puN mm/vmscan.c~ptrace-force-swap mm/vmscan.c
> --- lxc/mm/vmscan.c~ptrace-force-swap 2007-06-13 15:24:29.000000000 -0700
> +++ lxc-dave/mm/vmscan.c 2007-06-13 15:24:29.000000000 -0700
> @@ -441,167 +441,191 @@ cannot_free:
>   return 0;
>  }
>
> -/*
> - * shrink_page_list() returns the number of reclaimed pages
> - */
> -static unsigned long shrink_page_list(struct list_head *page_list,
> -     struct scan_control *sc)
> +int try_to_shrink_page(struct page *page, struct scan_control *sc)
>  {
> - LIST_HEAD(ret_pages);
> - struct pagevec freed_pvec;
> - int pgactivate = 0;
> - unsigned long nr_reclaimed = 0;
> + struct address_space *mapping;
> + int may_enter_fs;
> + int referenced;
>
> - cond_resched();
> + list_del(&page->lru);
```

>
> - pagevec_init(&freed_pvec, 1);
> - while (!list_empty(page_list)) {
> - struct address_space *mapping;
> - struct page *page;
> - int may_enter_fs;
> - int referenced;
> + if (TestSetPageLocked(page))
> + goto keep;
>
> - cond_resched();
> + VM_BUG_ON(PageActive(page));
>
> - page = lru_to_page(page_list);
> - list_del(&page->lru);
> + sc->nr_scanned++;
>
> - if (TestSetPageLocked(page))
> - goto keep;
> -
> - VM_BUG_ON(PageActive(page));
> + if (!sc->may_swap && page_mapped(page))
> + goto keep_locked;
>
> + /* Double the slab pressure for mapped and swapcache pages */
> + if (page_mapped(page) || PageSwapCache(page))
> sc->nr_scanned++;
>
> - if (!sc->may_swap && page_mapped(page))
> - goto keep_locked;
> -
> - /* Double the slab pressure for mapped and swapcache pages */
> - if (page_mapped(page) || PageSwapCache(page))
> - sc->nr_scanned++;
> + if (PageWriteback(page))
> + goto keep_locked;
>
> - if (PageWriteback(page))
> - goto keep_locked;
> -
> - referenced = page_referenced(page, 1);
> - /* In active use or really unfreeable? Activate it. */
> - if (referenced && page_mapping_inuse(page))
> - goto activate_locked;
> + referenced = page_referenced(page, 1);
> + /* In active use or really unfreeable? Activate it. */
> + if (referenced && page_mapping_inuse(page))
> + goto activate_locked;

```
>
>  #ifdef CONFIG_SWAP
> -  /*
> -   * Anonymous process memory has backing store?
> -   * Try to allocate it some swap space here.
> -   */
> -  if (PageAnon(page) && !PageSwapCache(page))
> -   if (!add_to_swap(page, GFP_ATOMIC))
> -    goto activate_locked;
> + /*
> +  * Anonymous process memory has backing store?
> +  * Try to allocate it some swap space here.
> +  */
> + if (PageAnon(page) && !PageSwapCache(page))
> +  if (!add_to_swap(page, GFP_ATOMIC))
> +   goto activate_locked;
>  #endif /* CONFIG_SWAP */
>
> -  mapping = page_mapping(page);
> -  may_enter_fs = (sc->gfp_mask & __GFP_FS) ||
> -   (PageSwapCache(page) && (sc->gfp_mask & __GFP_IO));
> + mapping = page_mapping(page);
> + may_enter_fs = (sc->gfp_mask & __GFP_FS) ||
> +  (PageSwapCache(page) && (sc->gfp_mask & __GFP_IO));
>
> -  /*
> -   * The page is mapped into the page tables of one or more
> -   * processes. Try to unmap it here.
> -   */
> -  if (page_mapped(page) && mapping) {
> -   switch (try_to_unmap(page, 0)) {
> -   case SWAP_FAIL:
> -    goto activate_locked;
> -   case SWAP_AGAIN:
> -    goto keep_locked;
> -   case SWAP_SUCCESS:
> -    ; /* try to free the page below */
> -   }
> + /*
> +  * The page is mapped into the page tables of one or more
> +  * processes. Try to unmap it here.
> +  */
> + if (page_mapped(page) && mapping) {
> +  switch (try_to_unmap(page, 0)) {
> +  case SWAP_FAIL:
> +   goto activate_locked;
> +  case SWAP_AGAIN:
> +   goto keep_locked;
```

```
> +  case SWAP_SUCCESS:
> +   ; /* try to free the page below */
>    }
> + }
>
> -  if (PageDirty(page)) {
> -   if (referenced)
> -    goto keep_locked;
> -   if (!may_enter_fs)
> -    goto keep_locked;
> -   if (!sc->may_writepage)
> -    goto keep_locked;
> + if (PageDirty(page)) {
> +  if (referenced)
> +   goto keep_locked;
> +  if (!may_enter_fs)
> +   goto keep_locked;
> +  if (!sc->may_writepage)
> +   goto keep_locked;
>
> -   /* Page is dirty, try to write it out here */
> -   switch(pageout(page, mapping)) {
> -   case PAGE_KEEP:
> +  /* Page is dirty, try to write it out here */
> +  switch(pageout(page, mapping)) {
> +  case PAGE_KEEP:
> +   goto keep_locked;
> +  case PAGE_ACTIVATE:
> +   goto activate_locked;
> +  case PAGE_SUCCESS:
> +   if (PageWriteback(page) || PageDirty(page))
> +    goto keep;
> +   /*
> +    * A synchronous write - probably a ramdisk.  Go
> +    * ahead and try to reclaim the page.
> +    */
> +   if (TestSetPageLocked(page))
> +    goto keep;
> +   if (PageDirty(page) || PageWriteback(page))
>      goto keep_locked;
> -   case PAGE_ACTIVATE:
> -    goto activate_locked;
> -   case PAGE_SUCCESS:
> -    if (PageWriteback(page) || PageDirty(page))
> -     goto keep;
> -    /*
> -     * A synchronous write - probably a ramdisk.  Go
> -     * ahead and try to reclaim the page.
```

```
> -     */
> -    if (TestSetPageLocked(page))
> -     goto keep;
> -    if (PageDirty(page) || PageWriteback(page))
> -     goto keep_locked;
> -    mapping = page_mapping(page);
> -   case PAGE_CLEAN:
> -    ; /* try to free the page below */
> -   }
> +   mapping = page_mapping(page);
> +  case PAGE_CLEAN:
> +   ; /* try to free the page below */
>    }
> + }
>
> -  /*
> -   * If the page has buffers, try to free the buffer mappings
> -   * associated with this page. If we succeed we try to free
> -   * the page as well.
> -   *
> -   * We do this even if the page is PageDirty().
> -   * try_to_release_page() does not perform I/O, but it is
> -   * possible for a page to have PageDirty set, but it is actually
> -   * clean (all its buffers are clean).  This happens if the
> -   * buffers were written out directly, with submit_bh(). ext3
> -   * will do this, as well as the blockdev mapping.
> -   * try_to_release_page() will discover that cleanness and will
> -   * drop the buffers and mark the page clean - it can be freed.
> -   *
> -   * Rarely, pages can have buffers and no ->mapping.  These are
> -   * the pages which were not successfully invalidated in
> -   * truncate_complete_page().  We try to drop those buffers here
> -   * and if that worked, and the page is no longer mapped into
> -   * process address space (page_count == 1) it can be freed.
> -   * Otherwise, leave the page on the LRU so it is swappable.
> -   */
> -  if (PagePrivate(page)) {
> -   if (!try_to_release_page(page, sc->gfp_mask))
> -    goto activate_locked;
> -   if (!mapping && page_count(page) == 1)
> -    goto free_it;
> -  }
> + /*
> +  * If the page has buffers, try to free the buffer mappings
> +  * associated with this page. If we succeed we try to free
> +  * the page as well.
> +  *
> +  * We do this even if the page is PageDirty().
```

```
> +  * try_to_release_page() does not perform I/O, but it is
> +  * possible for a page to have PageDirty set, but it is actually
> +  * clean (all its buffers are clean).  This happens if the
> +  * buffers were written out directly, with submit_bh(). ext3
> +  * will do this, as well as the blockdev mapping.
> +  * try_to_release_page() will discover that cleanness and will
> +  * drop the buffers and mark the page clean - it can be freed.
> +  *
> +  * Rarely, pages can have buffers and no ->mapping.  These are
> +  * the pages which were not successfully invalidated in
> +  * truncate_complete_page().  We try to drop those buffers here
> +  * and if that worked, and the page is no longer mapped into
> +  * process address space (page_count == 1) it can be freed.
> +  * Otherwise, leave the page on the LRU so it is swappable.
> +  */
> + if (PagePrivate(page)) {
> +  if (!try_to_release_page(page, sc->gfp_mask))
> +   goto activate_locked;
> +  if (!mapping && page_count(page) == 1)
> +   goto free_it;
> + }
>
> -  if (!mapping || !remove_mapping(mapping, page))
> -   goto keep_locked;
> + if (!mapping || !remove_mapping(mapping, page))
> +  goto keep_locked;
>
>  free_it:
> -  unlock_page(page);
> -  nr_reclaimed++;
> -  if (!pagevec_add(&freed_pvec, page))
> -   __pagevec_release_nonlru(&freed_pvec);
> -  continue;
> + unlock_page(page);
> + return 1;
>
>  activate_locked:
> -  SetPageActive(page);
> -  pgactivate++;
> + /* Not a candidate for swapping, so reclaim swap space. */
> + if (PageSwapCache(page) && vm_swap_full())
> +  remove_exclusive_swap_page(page);
> + SetPageActive(page);
> + count_vm_events(PGACTIVATE, 1);
>  keep_locked:
> -  unlock_page(page);
> + unlock_page(page);
>  keep:
```

```
> -  list_add(&page->lru, &ret_pages);
> -  VM_BUG_ON(PageLRU(page));
> + VM_BUG_ON(PageLRU(page));
> + return 0;
> +}
> +
> +/*
> + * shrink_page_list() returns the number of reclaimed pages
> + */
> +static unsigned long shrink_page_list(struct list_head *page_list,
> +      struct scan_control *sc)
> +{
> + LIST_HEAD(ret_pages);
> + struct pagevec freed_pvec;
> + unsigned long nr_reclaimed = 0;
> +
> + cond_resched();
> +
> + pagevec_init(&freed_pvec, 1);
> + while (!list_empty(page_list)) {
> +  struct page *page = lru_to_page(page_list);
> +  cond_resched();
> +  if (try_to_shrink_page(page, sc)) {
> +   nr_reclaimed++;
> +   if (!pagevec_add(&freed_pvec, page))
> +    __pagevec_release_nonlru(&freed_pvec);
> +  } else {
> +   list_add(&page->lru, &ret_pages);
> +  }
>    }
>    list_splice(&ret_pages, page_list);
>    if (pagevec_count(&freed_pvec))
>     __pagevec_release_nonlru(&freed_pvec);
> - count_vm_events(PGACTIVATE, pgactivate);
>    return nr_reclaimed;
>  }
>
> +int try_to_put_page_in_swap(struct page *page)
> +{
> +
> + get_page(page);
> + if (page_count(page) == 1)
> +  /* page was freed from under us. So we are done. */
> +  return -EAGAON;
> + lock_page(page);
> + if (PageWriteback(page))
> +  wait_on_page_writeback(page);
> + try_to_unmap(page, 0);
```

> + printk("page mapped: %d\n", page_mapped(page));
> + unlock_page(page);
> + put_page(page);
> + return 0;
> +}
> +
> /*
>   * zone->lru_lock is heavily contended. Some of the functions that
>   * shrink the lists perform better by taking out a batch of pages
> _
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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