Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Pavel Emelianov on Fri, 01 Jun 2007 07:47:18 GMT
View Forum Message <> Reply to Message

```
Balbir Singh wrote:
> Andrew Morton wrote:
>> On Wed, 30 May 2007 19:42:26 +0400
>> Pavel Emelianov <xemul@openvz.org> wrote:
>>
>>> Implement try_to_free_pages_in_container() to free the
>>> pages in container that has run out of memory.
>>>
>>> The scan_control->isolate_pages() function is set to
>>> isolate_pages_in_container() that isolates the container
>>> pages only. The exported __isolate_lru_page() call
>>> makes things look simpler than in the previous version.
>>>
>>> Includes fix from Balbir Singh <balbir@in.ibm.com>
>>>
>>>  }
>>>
>>> +void container_rss_move_lists(struct page *pg, bool active)
>>> +{
>>> + struct rss_container *rss;
>>> + struct page_container *pc;
>>> +
>>> + if (!page_mapped(pg))
>>> +  return;
>>> +
>>> + pc = page_container(pg);
>>> + if (pc == NULL)
>>> +  return;
>>> +
>>> + rss = pc->cnt;
>>> +
>>> + spin_lock(&rss->res.lock);
>>> + if (active)
>>> +  list_move(&pc->list, &rss->active_list);
>>> + else
>>> +  list_move(&pc->list, &rss->inactive_list);
>>> + spin_unlock(&rss->res.lock);
>>> +}
>> This is an interesting-looking function.  Please document it?
>>
>
> Will do. This function is called when we want to move a page in
> the LRU list. This could happen when a page is activated or when
> reclaim finds that a particular page cannot be reclaimed right now.
```

>
>> I'm inferring that the rss container has an active and inactive list and
>> that this basically follows the same operation as the traditional per-zone
>> lists?
>>
>
> Yes, correct.
>
>> Would I be correct in guessing that pages which are on the
>> per-rss-container lists are also eligible for reclaim off the traditional
>> page LRUs? If so, how does that work? When a page gets freed off the
>> per-zone LRUs does it also get removed from the per-rss_container LRU? But
>> how can this be right? Pages can get taken off the LRU and freed at
>> interrupt time, and this code isn't interrupt-safe.
>>
>
> Yes, before a page is reclaimed from the global LRU list, we go through
> page_remove_rmap() in try_to_unmap(). Pages are removed from the container
> LRU before they are reclaimed.

Yup. And this makes possible for additional charge.

>> I note that this lock is not irq-safe, whereas the lru locks are irq-safe.
>> So we don't perform the rotate_reclaimable_page() operation within the RSS
>> container? I think we could do so. I wonder if this was considered.
>>
>
> The lock needs to be interrupt safe.

Agree. I have found one place where we lost moving the page across
lists and this place is interrupt-accessible.

>> A description of how all this code works would help a lot.
>>

```
>>> +static unsigned long isolate_container_pages(unsigned long nr_to_scan,
>>> +  struct list_head *src, struct list_head *dst,
>>> +  unsigned long *scanned, struct zone *zone, int mode)
>>> +{
>>> + unsigned long nr_taken = 0;
>>> + struct page *page;
>>> + struct page_container *pc;
>>> + unsigned long scan;
>>> + LIST_HEAD(pc_list);
>>> +
>>> + for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>>> +  pc = list_entry(src->prev, struct page_container, list);
>>> +  page = pc->page;
>>> +  if (page_zone(page) != zone)
>>> +   continue;
>> That page_zone() check is interesting.  What's going on here?
>>
>> I'm suspecting that we have a problem here: if there are a lot of pages on
>> *src which are in the wrong zone, we can suffer reclaim distress leading to
>> omm-killings, or excessive CPU consumption?
>>
>
> We discussed this on lkml. Basically, now for every zone we try to reclaim
> pages from the container, it increases CPU utilization if we choose the wrong
> zone to reclaim from, but it provides the following benefit
>
> Code reuse (shrink_zone* is reused along with helper functions). I am not sure
> if Pavel had any other benefits in mind like benefits on a NUMA box.
```

Actually I planned to make RSS container look like a standalone machine
with active/inactive lists being per-zone/per-node. This will take all the
benefits from the current scanner.

Right now the lists are "global", but I'm looking at how zones are organized
to find out how to split them.

```
>>> + for_each_online_node(node) {
>>> +#ifdef CONFIG_HIGHMEM
>>> +  zones = NODE_DATA(node)->node_zonelists[ZONE_HIGHMEM].zones;
>>> +  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
>>> +   return 1;
>>> +#endif
>>> +  zones = NODE_DATA(node)->node_zonelists[ZONE_NORMAL].zones;
>>> +  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
>>> +   return 1;
>>> + }
>> Definitely need to handle ZONE_DMA32 and ZONE_DMA (some architectures put
>> all memory into ZONE_DMA (or they used to))
>>
>
> node_zonelists[ZONE_NORMAL].zones should contain ZONE_DMA and ZONE_DMA32 right?
```

Wait. Do you mean, that zones intersect with each other???