Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Pavel Emelianov on Thu, 31 May 2007 10:31:21 GMT
View Forum Message <> Reply to Message

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?
>
> 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 - each container tries to look like a zone, i.e. have two lists
and hold the pages there in LRU manner. The problem here (and it was
pointed out by you lower) is that these two lists store pages from
all the zones. This is still the largest TODO in this patchset. Sorry
for not pointing this out explicitly.

> 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

Yes. All the pages are accessible from booth - global and per-container
LRU lists and reclamation can be performed from booth.

> per-zone LRUs does it also get removed from the per-rss_container LRU? But
> how can this be right?

I don't get your idea here.

> Pages can get taken off the LRU and freed at
> interrupt time, and this code isn't interrupt-safe.

Actually, all the places we move the page within the container lists
are not in interrupts. But I have found at least one when the page
is being moved from one list to another, this place is IRQ-accessible,
but we don't move the page in container. Thus we have a BUG in
LRU maintenance but not in interrupt-safeness.

I will recheck this.

> 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.
>
> 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?

That's the TODO I have told above.