Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Andrew Morton on Wed, 30 May 2007 21:47:37 GMT
View Forum Message <> Reply to Message

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?

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.

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?

```
> +  list_move(&pc->list, &pc_list);
> +
> +  if (__isolate_lru_page(page, mode) == 0) {
> +   list_move(&page->lru, dst);
> +   nr_taken++;
> +  }
> + }
> +
> + list_splice(&pc_list, src);
> +
> + *scanned = scan;
> + return nr_taken;
> +}
> +
```

```
> +unsigned long isolate_pages_in_container(unsigned long nr_to_scan,
> +  struct list_head *dst, unsigned long *scanned,
> +  int order, int mode, struct zone *zone,
> +  struct rss_container *rss, int active)
> +{
> + unsigned long ret;
> +
> + spin_lock(&rss->res.lock);
> + if (active)
> +  ret = isolate_container_pages(nr_to_scan, &rss->active_list,
> +    dst, scanned, zone, mode);
> + else
> +  ret = isolate_container_pages(nr_to_scan, &rss->inactive_list,
> +    dst, scanned, zone, mode);
> + spin_unlock(&rss->res.lock);
> + return ret;
> +}
> +
>  void container_rss_add(struct page_container *pc)
>  {
>   struct page *pg;
>
> ...
>
> +#ifdef CONFIG_RSS_CONTAINER
> +unsigned long try_to_free_pages_in_container(struct rss_container *cnt)
> +{
> + struct scan_control sc = {
> +  .gfp_mask = GFP_KERNEL,
> +  .may_writepage = 1,
> +  .swap_cluster_max = 1,
> +  .may_swap = 1,
> +  .swappiness = vm_swappiness,
> +  .order = 0, /* in this case we wanted one page only */
> +  .cnt = cnt,
> +  .isolate_pages = isolate_pages_in_container,
> + };
> + int node;
> + struct zone **zones;
> +
> + 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))

> + return 0;
> +}
> +#endif
> +
> unsigned long try_to_free_pages(struct zone **zones, int order, gfp_t gfp_mask)
> {
>   struct scan_control sc = {