Subject: [PATCH 7/8] Per-container pages reclamation
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:41:58 GMT
View Forum Message <> Reply to Message

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>

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
--- ./include/linux/rss_container.h.rssreclaim 2007-06-04 12:21:11.000000000 +0400
+++ ./include/linux/rss_container.h 2007-06-04 12:22:38.000000000 +0400
@@ -77,6 +77,12 @@ void container_out_of_memory(struct rss_

 void mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
 void mm_free_container(struct mm_struct *mm);
+
+void container_rss_move_lists(struct page *pg, bool active);
+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 *, int active);
 #else
 static inline int container_rss_prepare(struct page *pg,
   struct vm_area_struct *vma, struct page_container **pc)
@@ -104,5 +110,8 @@ static inline void mm_init_container(str
 static inline void mm_free_container(struct mm_struct *mm)
 {
 }
+
+#define isolate_container_pages(nr, dst, scanned, rss, act, zone) ({ BUG(); 0;})
+#define container_rss_move_lists(pg, active) do { } while (0)
 #endif
 #endif
--- ./mm/rss_container.c.rssreclaim 2007-06-04 12:06:27.000000000 +0400
+++ ./mm/rss_container.c 2007-06-04 12:22:38.000000000 +0400
@@ -108,8 +108,16 @@ int container_rss_prepare(struct page *p
  if (pc == NULL)
   goto out_nomem;
```

```
- if (res_counter_charge(&rss->res, 1))
-  goto out_charge;
+ while (res_counter_charge(&rss->res, 1)) {
+  if (try_to_free_pages_in_container(rss)) {
+   atomic_inc(&rss->rss_reclaimed);
+   continue;
+  }
+
+  container_out_of_memory(rss);
+  if (test_thread_flag(TIF_MEMDIE))
+   goto out_charge;
+ }

  pc->page = page;
  pc->cnt = rss;
@@ -163,6 +171,95 @@ void container_rss_del(struct page_conta
 }

 /*
+ * reclamation code helpers
+ */
+
+/*
+ * Move the page across the active/inactive lists.
+ * It 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.  --balbir
+ */
+
+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;
+
+ /* we are called with zone->lru_lock held with irqs disabled */
+ 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);
+}
+
+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;
+  /*
+   * TODO: now we hold all the pages in one... ok, two lists
+   * and skip the pages from another zones with the check
+   * below.  this in not very good - try to make these lists
+   * per-zone to optimize this loop
+   */
+  if (page_zone(page) != zone)
+   continue;
+
+  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;
+
+ /* we are called with zone->lru_lock held with irqs disabled */
```

```
+ 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;
+}
+
+/*
  * interaction with the containers subsystem
  */

--- ./mm/swap.c.rssreclaim 2007-06-04 12:05:26.000000000 +0400
+++ ./mm/swap.c 2007-06-04 12:22:38.000000000 +0400
@@ -31,6 +31,7 @@
 #include <linux/cpu.h>
 #include <linux/notifier.h>
 #include <linux/init.h>
+#include <linux/rss_container.h>

 /* How many pages do we try to swap or page in/out together? */
 int page_cluster;
@@ -128,6 +129,7 @@ int rotate_reclaimable_page(struct page
  if (PageLRU(page) && !PageActive(page)) {
   list_move_tail(&page->lru, &zone->inactive_list);
   __count_vm_event(PGROTATED);
+  container_rss_move_lists(page, 0);
  }
  if (!test_clear_page_writeback(page))
   BUG();
@@ -148,6 +150,7 @@ void fastcall activate_page(struct page
   SetPageActive(page);
   add_page_to_active_list(zone, page);
   __count_vm_event(PGACTIVATE);
+  container_rss_move_lists(page, 1);
  }
  spin_unlock_irq(&zone->lru_lock);
 }
--- ./mm/vmscan.c.rssreclaim 2007-06-04 12:06:15.000000000 +0400
+++ ./mm/vmscan.c 2007-06-04 12:22:38.000000000 +0400
@@ -855,10 +855,13 @@ static unsigned long shrink_inactive_lis
    VM_BUG_ON(PageLRU(page));
    SetPageLRU(page);
    list_del(&page->lru);
-   if (PageActive(page))
```

```
+   if (PageActive(page)) {
     add_page_to_active_list(zone, page);
-   else
+    container_rss_move_lists(page, 1);
+   } else {
     add_page_to_inactive_list(zone, page);
+    container_rss_move_lists(page, 0);
+   }
    if (!pagevec_add(&pvec, page)) {
     spin_unlock_irq(&zone->lru_lock);
     __pagevec_release(&pvec);
@@ -1005,6 +1008,7 @@ force_reclaim_mapped:
   ClearPageActive(page);

   list_move(&page->lru, &zone->inactive_list);
+  container_rss_move_lists(page, 0);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
    __mod_zone_page_state(zone, NR_INACTIVE, pgmoved);
@@ -1033,6 +1037,7 @@ force_reclaim_mapped:
   SetPageLRU(page);
   VM_BUG_ON(!PageActive(page));
   list_move(&page->lru, &zone->active_list);
+  container_rss_move_lists(page, 1);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
    __mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
@@ -1272,6 +1277,41 @@ unsigned long try_to_free_pages(struct z
  return do_try_to_free_pages(zones, gfp_mask, &sc);
 }

+#ifdef CONFIG_RSS_CONTAINER
+/*
+ * user pages can be allocated from any zone starting from ZONE_HIGHMEM
+ * so try to shrink pages from them all
+ */
+#ifdef CONFIG_HIGHMEM
+#define ZONE_USERPAGES ZONE_HIGHMEM
+#else
+#define ZONE_USERPAGES ZONE_NORMAL
+#endif
+
+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 */
+  .container = cnt,
+  .isolate_pages = isolate_pages_in_container,
+ };
+ int node;
+ struct zone **zones;
+
+ for_each_online_node(node) {
+  zones = NODE_DATA(node)->node_zonelists[ZONE_USERPAGES].zones;
+  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+   return 1;
+ }
+ return 0;
+}
+#endif
+
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
```