Subject: [PATCH 7/8] Scanner changes needed to implement per-container scanner
Posted by Pavel Emelianov on Wed, 30 May 2007 15:35:20 GMT
View Forum Message <> Reply to Message

The core change is that the isolate_lru_pages() call is
replaced with struct scan_controll->isolate_pages() call.

Other changes include exporting __isolate_lru_page() for
per-container isolator and handling variable-to-pointer
changes in try_to_free_pages().

This makes possible to use different isolation routines
for per-container page reclamation. This will be used by
the following patch.

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

---

```
diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/swap.h
linux-2.6.22-rc2-mm1-0/include/linux/swap.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/swap.h 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-0/include/linux/swap.h 2007-05-30 16:13:09.000000000 +0400
@@ -192,6 +192,11 @@ extern void swap_setup(void);
 /* linux/mm/vmscan.c */
 extern unsigned long try_to_free_pages(struct zone **zones, int order,
     gfp_t gfp_mask);
+
+struct rss_container;
+extern unsigned long try_to_free_pages_in_container(struct rss_container *);
+int __isolate_lru_page(struct page *page, int mode);
+
 extern unsigned long shrink_all_memory(unsigned long nr_pages);
 extern int vm_swappiness;
 extern int remove_mapping(struct address_space *mapping, struct page *page);
diff -upr linux-2.6.22-rc2-mm1.orig/mm/vmscan.c linux-2.6.22-rc2-mm1-0/mm/vmscan.c
--- linux-2.6.22-rc2-mm1.orig/mm/vmscan.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-0/mm/vmscan.c 2007-05-30 16:13:09.000000000 +0400
@@ -47,6 +47,8 @@

 #include "internal.h"

+#include <linux/rss_container.h>
+
 struct scan_control {
  /* Incremented by the number of inactive pages that were scanned */
  unsigned long nr_scanned;
@@ -70,6 +72,13 @@ struct scan_control {
```

```
  int all_unreclaimable;

  int order;
+
+ struct rss_container *cnt;
+
+ unsigned long (*isolate_pages)(unsigned long nr, struct list_head *dst,
+    unsigned long *scanned, int order, int mode,
+    struct zone *zone, struct rss_container *cont,
+    int active);
 };

 #define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -622,7 +631,7 @@ keep:
  *
  * returns 0 on success, -ve errno on failure.
  */
-static int __isolate_lru_page(struct page *page, int mode)
+int __isolate_lru_page(struct page *page, int mode)
 {
  int ret = -EINVAL;

@@ -774,6 +783,19 @@ static unsigned long clear_active_flags(
  return nr_active;
 }

+static unsigned long isolate_pages_global(unsigned long nr,
+  struct list_head *dst, unsigned long *scanned,
+  int order, int mode, struct zone *z,
+  struct rss_container *cont, int active)
+{
+ if (active)
+  return isolate_lru_pages(nr, &z->active_list,
+    dst, scanned, order, mode);
+ else
+  return isolate_lru_pages(nr, &z->inactive_list,
+    dst, scanned, order, mode);
+}
+
 /*
  * shrink_inactive_list() is a helper for shrink_zone().  It returns the number
  * of reclaimed pages
@@ -797,11 +819,11 @@ static unsigned long shrink_inactive_lis
   unsigned long nr_freed;
   unsigned long nr_active;

-  nr_taken = isolate_lru_pages(sc->swap_cluster_max,
-        &zone->inactive_list,
```

```
-       &page_list, &nr_scan, sc->order,
-       (sc->order > PAGE_ALLOC_COSTLY_ORDER)?
-         ISOLATE_BOTH : ISOLATE_INACTIVE);
+  nr_taken = sc->isolate_pages(sc->swap_cluster_max, &page_list,
+    &nr_scan, sc->order,
+    (sc->order > PAGE_ALLOC_COSTLY_ORDER) ?
+     ISOLATE_BOTH : ISOLATE_INACTIVE,
+    zone, sc->cnt, 0);
    nr_active = clear_active_flags(&page_list);

   __mod_zone_page_state(zone, NR_ACTIVE, -nr_active);
@@ -950,8 +972,8 @@ force_reclaim_mapped:

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
- pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
-       &l_hold, &pgscanned, sc->order, ISOLATE_ACTIVE);
+ pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned,
+   sc->order, ISOLATE_ACTIVE, zone, sc->cnt, 1);
  zone->pages_scanned += pgscanned;
  __mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
  spin_unlock_irq(&zone->lru_lock);
@@ -1142,7 +1166,8 @@ static unsigned long shrink_zones(int pr
  * holds filesystem locks which prevent writeout this might not work, and the
  * allocation attempt will fail.
  */
-unsigned long try_to_free_pages(struct zone **zones, int order, gfp_t gfp_mask)
+unsigned long do_try_to_free_pages(struct zone **zones, gfp_t gfp_mask,
+  struct scan_control *sc)
 {
  int priority;
  int ret = 0;
@@ -1151,14 +1176,6 @@ unsigned long try_to_free_pages(struct z
  struct reclaim_state *reclaim_state = current->reclaim_state;
  unsigned long lru_pages = 0;
  int i;
- struct scan_control sc = {
-  .gfp_mask = gfp_mask,
-  .may_writepage = !laptop_mode,
-  .swap_cluster_max = SWAP_CLUSTER_MAX,
-  .may_swap = 1,
-  .swappiness = vm_swappiness,
-  .order = order,
- };

  delay_swap_prefetch();
  count_vm_event(ALLOCSTALL);
@@ -1174,17 +1191,18 @@ unsigned long try_to_free_pages(struct z
```

```
  }

  for (priority = DEF_PRIORITY; priority >= 0; priority--) {
-  sc.nr_scanned = 0;
+  sc->nr_scanned = 0;
   if (!priority)
    disable_swap_token();
-  nr_reclaimed += shrink_zones(priority, zones, &sc);
-  shrink_slab(sc.nr_scanned, gfp_mask, lru_pages);
+  nr_reclaimed += shrink_zones(priority, zones, sc);
+  if (sc->cnt == NULL)
+   shrink_slab(sc->nr_scanned, gfp_mask, lru_pages);
   if (reclaim_state) {
    nr_reclaimed += reclaim_state->reclaimed_slab;
    reclaim_state->reclaimed_slab = 0;
   }
-  total_scanned += sc.nr_scanned;
-  if (nr_reclaimed >= sc.swap_cluster_max) {
+  total_scanned += sc->nr_scanned;
+  if (nr_reclaimed >= sc->swap_cluster_max) {
    ret = 1;
    goto out;
   }
@@ -1196,18 +1214,18 @@ unsigned long try_to_free_pages(struct z
    * that's undesirable in laptop mode, where we *want* lumpy
    * writeout.  So in laptop mode, write out the whole world.
    */
-  if (total_scanned > sc.swap_cluster_max +
-     sc.swap_cluster_max / 2) {
+  if (total_scanned > sc->swap_cluster_max +
+     sc->swap_cluster_max / 2) {
    wakeup_pdflush(laptop_mode ? 0 : total_scanned);
-   sc.may_writepage = 1;
+   sc->may_writepage = 1;
   }

   /* Take a nap, wait for some writeback to complete */
-  if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+  if (sc->nr_scanned && priority < DEF_PRIORITY - 2)
    congestion_wait(WRITE, HZ/10);
  }
  /* top priority shrink_caches still had more to do? don't OOM, then */
- if (!sc.all_unreclaimable)
+ if (!sc->all_unreclaimable && sc->cnt == NULL)
   ret = 1;
 out:
  /*
@@ -1230,6 +1248,22 @@ out:
```

```
  return ret;
 }

+unsigned long try_to_free_pages(struct zone **zones, int order, gfp_t gfp_mask)
+{
+ struct scan_control sc = {
+  .gfp_mask = gfp_mask,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+  .order = order,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
+ };
+
+ return do_try_to_free_pages(zones, gfp_mask, &sc);
+}
+
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
@@ -1265,6 +1329,8 @@ static unsigned long balance_pgdat(pg_da
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
   .order = order,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  /*
   * temp_priority is used to remember the scanning priority at which
@@ -1604,6 +1670,8 @@ unsigned long shrink_all_memory(unsigned
   .swap_cluster_max = nr_pages,
   .may_writepage = 1,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };

  delay_swap_prefetch();
@@ -1789,6 +1857,8 @@ static int __zone_reclaim(struct zone *z
      SWAP_CLUSTER_MAX),
   .gfp_mask = gfp_mask,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  unsigned long slab_reclaimable;
```