Subject: [PATCH 7/8] Page scanner changes needed to implement per-container scanner
Posted by xemul on Mon, 09 Apr 2007 12:57:22 GMT
View Forum Message <> Reply to Message

Struct scan_control now carries:
* the RSS container to free pages in;
* pointer to an isolate_pages() function to isolate
  pages needed for the reclamation.

```
diff -upr linux-2.6.20.orig/mm/vmscan.c linux-2.6.20-2/mm/vmscan.c
--- linux-2.6.20.orig/mm/vmscan.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/vmscan.c 2007-04-09 11:26:06.000000000 +0400
@@ -45,6 +45,8 @@

 #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;
@@ -66,6 +68,10 @@ struct scan_control {
  int swappiness;

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

 /*
@@ -654,6 +660,17 @@ static unsigned long isolate_lru_pages(u
  return nr_taken;
 }

+static unsigned long isolate_pages_global(unsigned long nr,
+  struct list_head *dst, unsigned long *scanned,
+  struct zone *z, struct rss_container *cont,
+  int active)
+{
+ if (active)
+  return isolate_lru_pages(nr, &z->active_list, dst, scanned);
+ else
+  return isolate_lru_pages(nr, &z->inactive_list, dst, scanned);
+}
+
```

```
 /*
  * shrink_inactive_list() is a helper for shrink_zone().  It returns the number
  * of reclaimed pages
@@ -676,9 +693,9 @@ static unsigned long shrink_inactive_lis
   unsigned long nr_scan;
   unsigned long nr_freed;

-  nr_taken = isolate_lru_pages(sc->swap_cluster_max,
-          &zone->inactive_list,
-          &page_list, &nr_scan);
+  nr_taken = sc->isolate_pages(sc->swap_cluster_max, &page_list,
+    &nr_scan, zone, sc->cnt, 0);
+
   zone->nr_inactive -= nr_taken;
   zone->pages_scanned += nr_scan;
   spin_unlock_irq(&zone->lru_lock);
@@ -822,8 +839,8 @@ force_reclaim_mapped:

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
- pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
-        &l_hold, &pgscanned);
+ pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned, zone,
+   sc->cnt, 1);
  zone->pages_scanned += pgscanned;
  zone->nr_active -= pgmoved;
  spin_unlock_irq(&zone->lru_lock);
@@ -1012,7 +1031,9 @@ 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, gfp_t gfp_mask)
+
+static unsigned long do_try_to_free_pages(struct zone **zones, gfp_t gfp_mask,
+  struct scan_control *sc)
 {
  int priority;
  int ret = 0;
@@ -1021,13 +1042,6 @@ 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,
```

```
- };

  count_vm_event(ALLOCSTALL);

@@ -1041,17 +1055,18 @@ 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;
   }
@@ -1063,18 +1078,18 @@ 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)
   ret = 1;
 out:
  /*
@@ -1097,6 +1112,21 @@ out:
  return ret;
 }

+unsigned long try_to_free_pages(struct zone **zones, 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,
+  .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.
@@ -1131,6 +1190,8 @@ static unsigned long balance_pgdat(pg_da
   .may_swap = 1,
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  /*
   * temp_priority is used to remember the scanning priority at which
@@ -1436,6 +1497,8 @@ unsigned long shrink_all_memory(unsigned
   .swap_cluster_max = nr_pages,
   .may_writepage = 1,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };

  current->reclaim_state = &reclaim_state;
@@ -1619,6 +1682,8 @@ 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;
```