Subject: [PATCH -RSS 1/1] Fix reclaim failure
Posted by Balbir Singh on Mon, 04 Jun 2007 15:32:44 GMT
View Forum Message <> Reply to Message

This patch fixes the problem seen when a container goes over its limit, the
reclaim is unsuccessful and the application is terminated. The problem
is that all pages are by default added to the active list of the RSS
controller. When __isolate_lru_page() is called, it checks to see if
the list that the page is on (active or inactive) is the same as
what PageActive(page) returns. If this is not the case, the page is skipped.
In our case a page might not have the PG_active bit set and might be on
the active list of the container, thus we were ignoring those pages and
our reclaim fails, leading to the application being killed.

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>

```
 mm/rss_container.c |   31 +++++++++++++++++++++++--------
 1 file changed, 23 insertions(+), 8 deletions(-)

diff -puN mm/rss_container.c~rss-fix-free-of-active-pages mm/rss_container.c
--- linux-2.6.22-rc2-mm1/mm/rss_container.c~rss-fix-free-of-active-pages 2007-06-04
19:48:56.000000000 +0530
+++ linux-2.6.22-rc2-mm1-balbir/mm/rss_container.c 2007-06-04 19:50:18.000000000 +0530
@@ -205,18 +205,37 @@ void container_rss_move_lists(struct pag
 }

 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)
+  struct rss_container *rss, struct list_head *dst,
+  unsigned long *scanned, struct zone *zone, int mode,
+  int active)
 {
  unsigned long nr_taken = 0;
  struct page *page;
  struct page_container *pc;
  unsigned long scan;
  LIST_HEAD(pc_list);
+ struct list_head *src;
+
+  src = active ? &rss->active_list : &rss->inactive_list;

  for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
   pc = list_entry(src->prev, struct page_container, list);
   page = pc->page;
+
+  /*
+   * We might have got our active, inactive lists
+   * incorrect, fix it here
+   */
+  if (active && !PageActive(page)) {
+   list_move(&pc->list, &rss->inactive_list);
+   scan--;
+   continue;
+  } else if (!active && PageActive(page)) {
+   list_move(&pc->list, &rss->active_list);
+   scan--;
+   continue;
+  }
+
   /*
    * TODO: now we hold all the pages in one... ok, two lists
    * and skip the pages from another zones with the check
@@ -249,12 +268,8 @@ unsigned long isolate_pages_in_container

  /* 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);
+ ret = isolate_container_pages(nr_to_scan, rss, dst, scanned, zone,
+     mode, active);
  spin_unlock(&rss->res.lock);
  return ret;
 }
_
```

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

Subject: [PATCH -RSS 2/2] Fix limit check after reclaim
Posted by Balbir Singh on Mon, 04 Jun 2007 15:33:04 GMT
View Forum Message <> Reply to Message

This patch modifies the reclaim behaviour such that before calling the
container out of memory routine, it checks if as a result of the reclaim
(even though pages might not be fully reclaimed), the resident set size
of the container decreased before declaring the container as out of memory

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>

```
 include/linux/res_counter.h |   23 +++++++++++++++++++++++
 mm/rss_container.c          |   11 +++++++++++
 2 files changed, 34 insertions(+)

diff -puN mm/rss_container.c~rss-fix-limit-check-after-reclaim mm/rss_container.c
--- linux-2.6.22-rc2-mm1/mm/rss_container.c~rss-fix-limit-check-after-reclaim 2007-06-04
20:13:40.000000000 +0530
+++ linux-2.6.22-rc2-mm1-balbir/mm/rss_container.c 2007-06-04 20:13:40.000000000 +0530
@@ -114,6 +114,17 @@ int container_rss_prepare(struct page *p
    continue;
   }

+  /*
+   * try_to_free_pages() might not give us a full picture
+   * of reclaim. Some pages are reclaimed and might be moved
+   * to swap cache or just unmapped from the container.
+   * Check the limit again to see if the reclaim reduced the
+   * current usage of the container before calling the
+   * container OOM routine
+   */
+  if (res_counter_check_under_limit(&rss->res))
+   continue;
+
   container_out_of_memory(rss);
   if (test_thread_flag(TIF_MEMDIE))
    goto out_charge;
diff -puN include/linux/res_counter.h~rss-fix-limit-check-after-reclaim include/linux/res_counter.h
--- linux-2.6.22-rc2-mm1/include/linux/res_counter.h~rss-fix-limit-check-after-reclaim 2007-06-04
20:13:40.000000000 +0530
+++ linux-2.6.22-rc2-mm1-balbir/include/linux/res_counter.h 2007-06-04 20:15:46.000000000
+0530
@@ -99,4 +99,27 @@ int res_counter_charge(struct res_counte
 void res_counter_uncharge_locked(struct res_counter *cnt, unsigned long val);
 void res_counter_uncharge(struct res_counter *cnt, unsigned long val);

+static inline bool res_counter_limit_check_locked(struct res_counter *cnt)
+{
+ if (cnt->usage < cnt->limit)
+  return true;
+
+ return false;
+}
+
+/*
+ * Helper function to detect if the container is within it's limit or
+ * not. It's currently called from container_rss_prepare()
+ */
+static inline bool res_counter_check_under_limit(struct res_counter *cnt)
+{
+ bool ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ ret = res_counter_limit_check_locked(cnt);
+ spin_unlock_irqrestore(&cnt->lock, flags);
+ return ret;
+}
+
 #endif
diff -puN mm/vmscan.c~rss-fix-limit-check-after-reclaim mm/vmscan.c
_
```

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL