
Subject: [-mm PATCH 9/9] Memory controller make page_referenced() container aware (v4)

Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:11:19 GMT

[View Forum Message](#) <> [Reply to Message](#)

Make page_referenced() container aware. Without this patch, page_referenced() can cause a page to be skipped while reclaiming pages. This patch ensures that other containers do not hold pages in a particular container hostage. It is required to ensure that shared pages are freed from a container when they are not actively referenced from the container that brought them in

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

```
include/linux/memcontrol.h | 6 ++++++
include/linux/rmap.h       | 5 +++--
mm/memcontrol.c            | 5 ++++++
mm/rmap.c                  | 30 ++++++-----
mm/vmscan.c                | 4 ++--
5 files changed, 40 insertions(+), 10 deletions(-)
```

diff -puN include/linux/rmap.h~mem-control-per-container-page-referenced include/linux/rmap.h

linux-2.6.23-rc1-mm1/include/linux/rmap.h~mem-control-per-container-page-referenced 2007-07-28 01:12:50.000000000 +0530

+++ linux-2.6.23-rc1-mm1-balbir/include/linux/rmap.h 2007-07-28 01:12:50.000000000 +0530

@@ -8,6 +8,7 @@

#include <linux/slab.h>

#include <linux/mm.h>

#include <linux/spinlock.h>

+#include <linux/memcontrol.h>

/*

* The anon_vma heads a list of private "related" vmas, to scan if

@@ -86,7 +87,7 @@ static inline void page_dup_rmap(struct

/*

* Called from mm/vmscan.c to handle paging out

*/

-int page_referenced(struct page *, int is_locked);

+int page_referenced(struct page *, int is_locked, struct mem_container *cnt);

int try_to_unmap(struct page *, int ignore_refs);

/*

@@ -114,7 +115,7 @@ int page_mkclean(struct page *);

#define anon_vma_prepare(vma) (0)

#define anon_vma_link(vma) do {} while (0)

```

-#define page_referenced(page,l) TestClearPageReferenced(page)
+#define page_referenced(page,l,cnt) TestClearPageReferenced(page)
#define try_to_unmap(page, refs) SWAP_FAIL

static inline int page_mkclean(struct page *page)
diff -puN mm/rmap.c~mem-control-per-container-page-referenced mm/rmap.c
--- linux-2.6.23-rc1-mm1/mm/rmap.c~mem-control-per-container-page-referenced 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/rmap.c 2007-07-28 01:12:50.000000000 +0530
@@ -298,7 +298,8 @@ out:
    return referenced;
}

-static int page_referenced_anon(struct page *page)
+static int page_referenced_anon(struct page *page,
+    struct mem_container *mem_cont)
{
    unsigned int mapcount;
    struct anon_vma *anon_vma;
@@ -311,6 +312,13 @@ static int page_referenced_anon(struct p

    mapcount = page_mapcount(page);
    list_for_each_entry(vma, &anon_vma->head, anon_vma_node) {
+ /*
+  * If we are reclaiming on behalf of a container, skip
+  * counting on behalf of references from different
+  * containers
+  */
+ if (mem_cont && (mm_container(vma->vm_mm) != mem_cont))
+ continue;
    referenced += page_referenced_one(page, vma, &mapcount);
    if (!mapcount)
        break;
@@ -331,7 +339,8 @@ static int page_referenced_anon(struct p
*
* This function is only called from page_referenced for object-based pages.
*/
-static int page_referenced_file(struct page *page)
+static int page_referenced_file(struct page *page,
+    struct mem_container *mem_cont)
{
    unsigned int mapcount;
    struct address_space *mapping = page->mapping;
@@ -364,6 +373,13 @@ static int page_referenced_file(struct p
    mapcount = page_mapcount(page);

    vma_prio_tree_foreach(vma, &iter, &mapping->i_mmap, pgoff, pgoff) {

```

```

+ /*
+  * If we are reclaiming on behalf of a container, skip
+  * counting on behalf of references from different
+  * containers
+  */
+ if (mem_cont && (mm_container(vma->vm_mm) != mem_cont))
+ continue;
+ if ((vma->vm_flags & (VM_LOCKED|VM_MAYSHARE))
+     == (VM_LOCKED|VM_MAYSHARE)) {
+     referenced++;
@@ -386,7 +402,8 @@ static int page_referenced_file(struct p
+ * Quick test_and_clear_referenced for all mappings to a page,
+ * returns the number of ptes which referenced the page.
+ */
-int page_referenced(struct page *page, int is_locked)
+int page_referenced(struct page *page, int is_locked,
+ struct mem_container *mem_cont)
{
    int referenced = 0;

@@ -398,14 +415,15 @@ int page_referenced(struct page *page, i

    if (page_mapped(page) && page->mapping) {
        if (PageAnon(page))
- referenced += page_referenced_anon(page);
+ referenced += page_referenced_anon(page, mem_cont);
        else if (is_locked)
- referenced += page_referenced_file(page);
+ referenced += page_referenced_file(page, mem_cont);
        else if (TestSetPageLocked(page))
            referenced++;
        else {
            if (page->mapping)
- referenced += page_referenced_file(page);
+ referenced +=
+ page_referenced_file(page, mem_cont);
            unlock_page(page);
        }
    }
diff -puN mm/vmscan.c~mem-control-per-container-page-referenced mm/vmscan.c
--- linux-2.6.23-rc1-mm1/mm/vmscan.c~mem-control-per-container-page-referenced 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/vmscan.c 2007-07-28 01:12:50.000000000 +0530
@@ -479,7 +479,7 @@ static unsigned long shrink_page_list(st
    if (PageWriteback(page))
        goto keep_locked;

- referenced = page_referenced(page, 1);

```

```

+ referenced = page_referenced(page, 1, sc->mem_container);
/* In active use or really unfreeable? Activate it. */
if (sc->order <= PAGE_ALLOC_COSTLY_ORDER &&
    referenced && page_mapping_inuse(page))
@@ -971,7 +971,7 @@ force_reclaim_mapped:
    if (page_mapped(page)) {
        if (!reclaim_mapped ||
            (total_swap_pages == 0 && PageAnon(page)) ||
-        page_referenced(page, 0)) {
+        page_referenced(page, 0, sc->mem_container)) {
            list_add(&page->lru, &l_active);
            continue;
        }
diff -puN include/linux/memcontrol.h~mem-control-per-container-page-referenced
include/linux/memcontrol.h
---
linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-per-container-page-referenced 20
07-07-28 01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:50.000000000
+0530
@@ -42,6 +42,7 @@ extern unsigned long mem_container_isola
    int active);
extern void mem_container_out_of_memory(struct mem_container *mem);
extern int mem_container_cache_charge(struct page *page, struct mm_struct *mm);
+extern struct mem_container *mm_container(struct mm_struct *mm);

static inline void mem_container_uncharge_page(struct page *page)
{
@@ -91,6 +92,11 @@ static inline int mem_container_cache_ch
    return 0;
}

+static inline struct mem_container *mm_container(struct mm_struct *mm)
+{
+ return NULL;
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */

#endif /* _LINUX_MEMCONTROL_H */
diff -puN mm/memcontrol.c~mem-control-per-container-page-referenced mm/memcontrol.c
---
linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-per-container-page-referenced 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530
@@ -100,6 +100,11 @@ struct mem_container *mem_container_from
    struct mem_container, css);
}

```

```
+inline struct mem_container *mm_container(struct mm_struct *mm)
+{
+ return rcu_dereference(mm->mem_container);
+}
+
+void mm_init_container(struct mm_struct *mm, struct task_struct *p)
+{
+ struct mem_container *mem;
```

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
