

---

Subject: [PATCH 31/33] memory controller make page\_referenced container aware v7

Posted by [Paul Menage](#) on Mon, 17 Sep 2007 21:03:38 GMT

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

---

From: Balbir Singh <balbir@linux.vnet.ibm.com>  
(container->cgroup renaming by Paul Menage <menage@google.com>)

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

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

Signed-off-by: Paul Menage <menage@google.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/memcontrol.h~memory-controller-make-page\_referenced-cgroup-aware-v7

include/linux/memcontrol.h

--- a/include/linux/memcontrol.h~memory-controller-make-page\_referenced-cgroup-aware-v7

+++ a/include/linux/memcontrol.h

@ @ -43,6 +43,7 @ @ extern unsigned long mem\_cgroup\_isola  
int active);

extern void mem\_cgroup\_out\_of\_memory(struct mem\_cgroup \*mem);  
extern int mem\_cgroup\_cache\_charge(struct page \*page, struct mm\_struct \*mm);  
+extern struct mem\_cgroup \*mm\_cgroup(struct mm\_struct \*mm);

static inline void mem\_cgroup\_uncharge\_page(struct page \*page)  
{  
@ @ -93,6 +94,11 @ @ static inline int mem\_cgroup\_cache\_ch  
return 0;  
}

+static inline struct mem\_cgroup \*mm\_cgroup(struct mm\_struct \*mm)  
+{  
+ return NULL;  
+}  
+  
#endif /\* CONFIG\_CGROUP\_MEM\_CONT \*/

```

#endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/rmap.h~memory-controller-make-page_referenced-cgroup-aware-v7
include/linux/rmap.h
--- a/include/linux/rmap.h~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/include/linux/rmap.h
@@ -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_cgroup *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/memcontrol.c~memory-controller-make-page_referenced-cgroup-aware-v7
mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/mm/memcontrol.c
@@ -109,6 +109,11 @@ struct mem_cgroup *mem_cgroup_from
    struct mem_cgroup, css);
}

+inline struct mem_cgroup *mm_cgroup(struct mm_struct *mm)
+{
+ return rcu_dereference(mm->mem_cgroup);
+}
+
void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
{
    struct mem_cgroup *mem;
diff -puN mm/rmap.c~memory-controller-make-page_referenced-cgroup-aware-v7 mm/rmap.c

```

```

--- a/mm/rmap.c~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/mm/rmap.c
@@ -299,7 +299,8 @@ out:
    return referenced;
}

-static int page_referenced_anon(struct page *page)
+static int page_referenced_anon(struct page *page,
+ struct mem_cgroup *mem_cont)
{
    unsigned int mapcount;
    struct anon_vma *anon_vma;
@@ -312,6 +313,13 @@ 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 cgroup, skip
+  * counting on behalf of references from different
+  * cgroups
+  */
+ if (mem_cont && (mm_cgroup(vma->vm_mm) != mem_cont))
+ continue;
    referenced += page_referenced_one(page, vma, &mapcount);
    if (!mapcount)
        break;
@@ -332,7 +340,8 @@ 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_cgroup *mem_cont)
{
    unsigned int mapcount;
    struct address_space *mapping = page->mapping;
@@ -365,6 +374,13 @@ 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 cgroup, skip
+  * counting on behalf of references from different
+  * cgroups
+  */
+ if (mem_cont && (mm_cgroup(vma->vm_mm) != mem_cont))
+ continue;
    if ((vma->vm_flags & (VM_LOCKED|VM_MAYSHARE))

```

```

    == (VM_LOCKED|VM_MAYSHARE)) {
        referenced++;
@@ -387,7 +403,8 @@ 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_cgroup *mem_cont)
{
    int referenced = 0;

@@ -399,14 +416,15 @@ 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~memory-controller-make-page_referenced-cgroup-aware-v7
mm/vmscan.c
--- a/mm/vmscan.c~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/mm/vmscan.c
@@ -503,7 +503,7 @@ static unsigned long shrink_page_list(st
    goto keep_locked;
}

- referenced = page_referenced(page, 1);
+ referenced = page_referenced(page, 1, sc->mem_cgroup);
/* In active use or really unfreeable? Activate it. */
if (sc->order <= PAGE_ALLOC_COSTLY_ORDER &&
    referenced && page_mapping_inuse(page))
@@ -1057,7 +1057,7 @@ 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_cgroup)) {  
    list_add(&page->lru, &l_active);  
    continue;  
}
```

—

--

---

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

---