

---

Subject: [-mm PATCH 6/7] Memory controller add per container LRU and reclaim  
Posted by [Balbir Singh](#) on Wed, 04 Jul 2007 22:22:34 GMT

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

---

Add the meta\_page to the per container LRU. The reclaim algorithm has been modified to make the isolate\_lru\_pages() as a pluggable component. The scan\_control data structure now accepts the container on behalf of which reclaims are carried out. try\_to\_free\_pages() has been extended to become container aware.

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

---

```
include/linux/memcontrol.h | 11 +++
include/linux/res_counter.h | 23 ++++++++
include/linux/swap.h       | 3 +
mm/memcontrol.c           | 88 ++++++++
mm/swap.c                 | 2
mm/vmscan.c               | 129 ++++++++
6 files changed, 230 insertions(+), 26 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-lru-and-reclaim include/linux/memcontrol.h
```

```
--- linux-2.6.22-rc6/include/linux/memcontrol.h~mem-control-lru-and-reclaim 2007-07-04
```

```
15:05:31.000000000 -0700
```

```
+++ linux-2.6.22-rc6-balbir/include/linux/memcontrol.h 2007-07-04 15:05:31.000000000 -0700
```

```
@@ -26,6 +26,13 @@ extern void page_assign_meta_page(struct
extern struct meta_page *page_get_meta_page(struct page *page);
extern int mem_container_charge(struct page *page, struct mm_struct *mm);
extern void mem_container_uncharge(struct meta_page *mp);
+extern void mem_container_move_lists(struct meta_page *mp, bool active);
+extern unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
+ struct list_head *dst,
+ unsigned long *scanned, int order,
+ int mode, struct zone *z,
+ struct mem_container *mem_cont,
+ int active);
```

```
#else /* CONFIG_CONTAINER_MEM_CONT */
static inline void mm_init_container(struct mm_struct *mm,
@@ -56,6 +63,10 @@ static inline void mem_container_uncharg
{
}
```

```
+static inline void mem_container_move_lists(struct meta_page *mp, bool active)
+{
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */
```

```

#endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/swap.h~mem-control-lru-and-reclaim include/linux/swap.h
--- linux-2.6.22-rc6/include/linux/swap.h~mem-control-lru-and-reclaim 2007-07-04
15:05:31.000000000 -0700
+++ linux-2.6.22-rc6-balbir/include/linux/swap.h 2007-07-04 15:05:31.000000000 -0700
@@ -6,6 +6,7 @@
#include <linux/mmzone.h>
#include <linux/list.h>
#include <linux/sched.h>
+#include <linux/memcontrol.h>

#include <asm/atomic.h>
#include <asm/page.h>
@@ -191,6 +192,8 @@ 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);
+extern unsigned long try_to_free_mem_container_pages(struct mem_container *mem);
+extern 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 -puN mm/memcontrol.c~mem-control-lru-and-reclaim mm/memcontrol.c
--- linux-2.6.22-rc6/mm/memcontrol.c~mem-control-lru-and-reclaim 2007-07-04
15:05:31.000000000 -0700
+++ linux-2.6.22-rc6-balbir/mm/memcontrol.c 2007-07-04 15:05:31.000000000 -0700
@@ -19,6 +19,8 @@
#include <linux/page-flags.h>
#include <linux/bit_spinlock.h>
#include <linux/rcupdate.h>
+#include <linux/swap.h>
+#include <linux/spinlock.h>

struct container_subsys mem_container_subsys;

@@ -103,6 +105,71 @@ void __always_inline unlock_meta_page(st
    bit_spin_unlock(PG_metapage, &page->flags);
}

+unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
+    struct list_head *dst,
+    unsigned long *scanned, int order,
+    int mode, struct zone *z,
+    struct mem_container *mem_cont,
+    int active)
+{
+    unsigned long nr_taken = 0;

```

```

+ struct page *page;
+ unsigned long scan;
+ LIST_HEAD(mp_list);
+ struct list_head *src;
+ struct meta_page *mp;
+
+ if (active)
+ src = &mem_cont->active_list;
+ else
+ src = &mem_cont->inactive_list;
+
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+ mp = list_entry(src->prev, struct meta_page, list);
+ page = mp->page;
+
+ if (PageActive(page) && !active) {
+ mem_container_move_lists(mp, true);
+ scan--;
+ continue;
+ }
+ if (!PageActive(page) && active) {
+ mem_container_move_lists(mp, false);
+ scan--;
+ continue;
+ }
+
+ /*
+  * Reclaim, per zone
+  * TODO: make the active/inactive lists per zone
+  */
+ if (page_zone(page) != z)
+ continue;
+
+ list_move(&mp->list, &mp_list);
+ if (__isolate_lru_page(page, mode) == 0) {
+ list_move(&page->lru, dst);
+ nr_taken++;
+ }
+ }
+
+ list_splice(&mp_list, src);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+/*
+ * This routine assumes that the appropriate zone's lru lock is already held

```

```

+ */
+void mem_container_move_lists(struct meta_page *mp, bool active)
+{
+ if (active)
+ list_move(&mp->list, &mp->mem_container->active_list);
+ else
+ list_move(&mp->list, &mp->mem_container->inactive_list);
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
@@ -159,8 +226,22 @@ int mem_container_charge(struct page *pa
+ * If we created the meta_page, we should free it on exceeding
+ * the container limit.
+ */
- if (res_counter_charge(&mem->res, 1))
+ while (res_counter_charge(&mem->res, 1)) {
+ if (try_to_free_mem_container_pages(mem))
+ continue;
+
+ /*
+ * try_to_free_mem_container_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 giving up
+ */
+ if (res_counter_check_under_limit(&mem->res))
+ continue;
+
+ goto free_mp;
+ }

lock_meta_page(page);
/*
@@ -177,6 +258,8 @@ int mem_container_charge(struct page *pa
mp->page = page;
page_assign_meta_page(page, mp);

+ list_add(&mp->list, &mem->active_list);
+
done:
unlock_meta_page(page);
return 0;
@@ -205,12 +288,15 @@ void mem_container_uncharge(struct meta_

if (atomic_dec_and_test(&mp->ref_cnt)) {

```

```

page = mp->page;
+
lock_meta_page(page);
mem = mp->mem_container;
css_put(&mem->css);
page_assign_meta_page(page, NULL);
unlock_meta_page(page);
+
res_counter_uncharge(&mem->res, 1);
+ list_del(&mp->list);
kfree(mp);
}
}

```

diff -puN mm/swap.c~mem-control-lru-and-reclaim mm/swap.c

--- linux-2.6.22-rc6/mm/swap.c~mem-control-lru-and-reclaim 2007-07-04 15:05:31.000000000  
-0700

+++ linux-2.6.22-rc6-balbir/mm/swap.c 2007-07-04 15:05:31.000000000 -0700

@@ -31,6 +31,7 @@

#include <linux/cpu.h>

#include <linux/notifier.h>

#include <linux/init.h>

+#include <linux/memcontrol.h>

/\* How many pages do we try to swap or page in/out together? \*/

int page\_cluster;

@@ -148,6 +149,7 @@ void fastcall activate\_page(struct page

SetPageActive(page);

add\_page\_to\_active\_list(zone, page);

\_\_count\_vm\_event(PGACTIVATE);

+ mem\_container\_move\_lists(page\_get\_meta\_page(page), true);

}

spin\_unlock\_irq(&zone->lru\_lock);

}

diff -puN mm/vmscan.c~mem-control-lru-and-reclaim mm/vmscan.c

--- linux-2.6.22-rc6/mm/vmscan.c~mem-control-lru-and-reclaim 2007-07-04 15:05:31.000000000  
-0700

+++ linux-2.6.22-rc6-balbir/mm/vmscan.c 2007-07-04 15:05:31.000000000 -0700

@@ -39,6 +39,7 @@

#include <linux/delay.h>

#include <linux/kthread.h>

#include <linux/freezer.h>

+#include <linux/memcontrol.h>

#include <asm/tlbflush.h>

#include <asm/div64.h>

@@ -70,6 +71,15 @@ struct scan\_control {

int all\_unreclaimable;

```

int order;
+
+ /* Which container do we reclaim from */
+ struct mem_container *mem_container;
+
+ /* Pluggable isolate pages callback */
+ unsigned long (*isolate_pages)(unsigned long nr, struct list_head *dst,
+ unsigned long *scanned, int order, int mode,
+ struct zone *z, struct mem_container *mem_cont,
+ int active);
};

#define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -604,7 +614,7 @@ 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;

@@ -738,6 +748,21 @@ 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, int order,
+ int mode, struct zone *z,
+ struct mem_container *mem_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);
+}
+
+/*
+ * clear_active_flags() is a helper for shrink_active_list(), clearing
+ * any active bits from the pages in the list.
@@ -779,11 +804,11 @@ 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,
+ 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);
+   ISOLATE_BOTH : ISOLATE_INACTIVE,
+   zone, sc->mem_container, 0);
nr_active = clear_active_flags(&page_list);

__mod_zone_page_state(zone, NR_ACTIVE, -nr_active);
@@ -932,8 +957,9 @@ 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->mem_container, 1);
zone->pages_scanned += pgscanned;
__mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
spin_unlock_irq(&zone->lru_lock);
@@ -947,10 +973,14 @@ force_reclaim_mapped:
    (total_swap_pages == 0 && PageAnon(page)) ||
    page_referenced(page, 0) {
    list_add(&page->lru, &l_active);
+   mem_container_move_lists(
+   page_get_meta_page(page), true);
    continue;
    }
    } else if (TestClearPageReferenced(page)) {
    list_add(&page->lru, &l_active);
+   mem_container_move_lists(page_get_meta_page(page),
+   true);
    continue;
    }
    list_add(&page->lru, &l_inactive);
@@ -968,6 +998,7 @@ force_reclaim_mapped:
    ClearPageActive(page);

    list_move(&page->lru, &zone->inactive_list);
+   mem_container_move_lists(page_get_meta_page(page), false);
    pgmoved++;
    if (!pagevec_add(&pvec, page)) {
    __mod_zone_page_state(zone, NR_INACTIVE, pgmoved);
@@ -996,6 +1027,7 @@ force_reclaim_mapped:
    SetPageLRU(page);

```

```

VM_BUG_ON(!PageActive(page));
list_move(&page->lru, &zone->active_list);
+ mem_container_move_lists(page_get_meta_page(page), true);
pgmoved++;
if (!pagevec_add(&pvec, page)) {
    __mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
@@ -1127,7 +1159,8 @@ 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;
@@ -1136,14 +1169,6 @@ 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);
@@ -1159,17 +1184,22 @@ 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);
+ /*
+ * Don't shrink slabs when reclaiming memory from
+ * over limit containers
+ */
+ if (sc->mem_container == 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;
}
@@ -1181,18 +1211,18 @@ 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->mem_container == NULL)
    ret = 1;
out:
/*
@@ -1215,6 +1245,53 @@ 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,

```

```

+ .mem_container = NULL,
+ .isolate_pages = isolate_pages_global,
+ };
+
+ return do_try_to_free_pages(zones, gfp_mask, &sc);
+}
+
+#ifdef CONFIG_CONTAINER_MEM_CONT
+
+#ifdef CONFIG_HIGHMEM
+#define ZONE_USERPAGES ZONE_HIGHMEM
+#else
+#define ZONE_USERPAGES ZONE_NORMAL
+#endif
+
+unsigned long try_to_free_mem_container_pages(struct mem_container *mem_cont)
+{
+ struct scan_control sc = {
+ .gfp_mask = GFP_KERNEL,
+ .may_swap = 1,
+ .swap_cluster_max = SWAP_CLUSTER_MAX,
+ .swappiness = vm_swappiness,
+ .order = 1,
+ .mem_container = mem_cont,
+ .isolate_pages = mem_container_isolate_pages,
+ };
+ int node;
+ struct zone **zones;
+
+ for_each_online_node(node) {
+ zones = NODE_DATA(node)->node_zonelist[ZONE_USERPAGES].zones;
+ if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+ return 1;
+ }
+ return 0;
+}
+#endif
+
+/*
+ * For kswapd, balance_pgdat() will work across all this node's zones until
+ * they are all at pages_high.
+ @@ -1250,6 +1327,8 @@ static unsigned long balance_pgdat(pg_da
+ .swap_cluster_max = SWAP_CLUSTER_MAX,
+ .swappiness = vm_swappiness,
+ .order = order,
+ .mem_container = NULL,
+ .isolate_pages = isolate_pages_global,
+ };

```

```

/*
 * temp_priority is used to remember the scanning priority at which
diff -puN include/linux/res_counter.h~mem-control-lru-and-reclaim include/linux/res_counter.h
--- linux-2.6.22-rc6/include/linux/res_counter.h~mem-control-lru-and-reclaim 2007-07-04
15:05:31.000000000 -0700
+++ linux-2.6.22-rc6-balbir/include/linux/res_counter.h 2007-07-04 15:05:31.000000000 -0700
@@ -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

```

--  
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>

---