
Subject: [PATCH 25/29] memory controller add per container lru and reclaim v7
Posted by [Paul Menage](#) on Tue, 11 Sep 2007 19:53:04 GMT

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

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Add the page_cgroup to the per cgroup 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 cgroup on behalf of which reclaims are carried out. try_to_free_pages() has been extended to become cgroup aware.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

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

Cc: Paul Menage <menage@google.com>

Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>

Cc: "Eric W. Biederman" <ebiederm@xmission.com>

Cc: Nick Piggin <nickpiggin@yahoo.com.au>

Cc: Kirill Korotaev <dev@sw.ru>

Cc: Herbert Poetzl <herbert@13thfloor.at>

Cc: David Rientjes <rientjes@google.com>

Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>

Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
include/linux/memcontrol.h | 12 +++
include/linux/res_counter.h | 23 +++++
include/linux/swap.h       | 3
mm/memcontrol.c            | 135 +++++
mm/swap.c                  | 2
mm/vmscan.c                | 126 +++++
6 files changed, 275 insertions(+), 26 deletions(-)
```

diff -puN include/linux/memcontrol.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7

include/linux/memcontrol.h

--- a/include/linux/memcontrol.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7

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

```
@@ -32,6 +32,13 @@ extern void page_assign_page_cgroup(s
extern struct page_cgroup *page_get_page_cgroup(struct page *page);
extern int mem_cgroup_charge(struct page *page, struct mm_struct *mm);
extern void mem_cgroup_uncharge(struct page_cgroup *pc);
+extern void mem_cgroup_move_lists(struct page_cgroup *pc, bool active);
+extern unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
+ struct list_head *dst,
+ unsigned long *scanned, int order,
+ int mode, struct zone *z,
+ struct mem_cgroup *mem_cont,
+ int active);
```

```

static inline void mem_cgroup_uncharge_page(struct page *page)
{
@@ -71,6 +78,11 @@ static inline void mem_cgroup_uncharg
{
}

+static inline void mem_cgroup_move_lists(struct page_cgroup *pc,
+    bool active)
+{
+}
+
#endif /* CONFIG_CGROUP_MEM_CONT */

#endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/res_counter.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
include/linux/res_counter.h
--- a/include/linux/res_counter.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/include/linux/res_counter.h
@@ -99,4 +99,27 @@ int res_counter_charge(struct res_counte
void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
void res_counter_uncharge(struct res_counter *counter, 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 cgroup is within it's limit or
+ * not. It's currently called from cgroup_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 include/linux/swap.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7

```

```

include/linux/swap.h
--- a/include/linux/swap.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/include/linux/swap.h
@@ -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>
@@ -190,6 +191,8 @@ 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_cgroup_pages(struct mem_cgroup *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~memory-controller-add-per-cgroup-lru-and-reclaim-v7
mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/mm/memcontrol.c
@@ -24,8 +24,12 @@
#include <linux/page-flags.h>
#include <linux/bit_spinlock.h>
#include <linux/rcupdate.h>
+#include <linux/swap.h>
+#include <linux/spinlock.h>
+#include <linux/fs.h>

struct cgroup_subsys mem_cgroup_subsys;
+static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

/*
 * The memory controller data structure. The memory controller controls both
@@ -51,6 +55,10 @@ struct mem_cgroup {
 */
    struct list_head active_list;
    struct list_head inactive_list;
+ /*
+  * spin_lock to protect the per cgroup LRU
+  */
+ spinlock_t lru_lock;
};

/*

```

```

@@ -141,6 +149,94 @@ void __always_inline unlock_page_contain
    bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
}

+void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
+{
+ if (active)
+  list_move(&pc->lru, &pc->mem_cgroup->active_list);
+ else
+  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+}
+
+/*
+ * This routine assumes that the appropriate zone's lru lock is already held
+ */
+void mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
+{
+ struct mem_cgroup *mem;
+ if (!pc)
+  return;
+
+ mem = pc->mem_cgroup;
+
+ spin_lock(&mem->lru_lock);
+ __mem_cgroup_move_lists(pc, active);
+ spin_unlock(&mem->lru_lock);
+}
+
+unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
+   struct list_head *dst,
+   unsigned long *scanned, int order,
+   int mode, struct zone *z,
+   struct mem_cgroup *mem_cont,
+   int active)
+{
+ unsigned long nr_taken = 0;
+ struct page *page;
+ unsigned long scan;
+ LIST_HEAD(pc_list);
+ struct list_head *src;
+ struct page_cgroup *pc;
+
+ if (active)
+  src = &mem_cont->active_list;
+ else
+  src = &mem_cont->inactive_list;
+
+ spin_lock(&mem_cont->lru_lock);

```

```

+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+   pc = list_entry(src->prev, struct page_cgroup, lru);
+   page = pc->page;
+   VM_BUG_ON(!pc);
+
+   if (PageActive(page) && !active) {
+     __mem_cgroup_move_lists(pc, true);
+     scan--;
+     continue;
+   }
+   if (!PageActive(page) && active) {
+     __mem_cgroup_move_lists(pc, false);
+     scan--;
+     continue;
+   }
+
+   /*
+    * Reclaim, per zone
+    * TODO: make the active/inactive lists per zone
+    */
+   if (page_zone(page) != z)
+     continue;
+
+   /*
+    * Check if the meta page went away from under us
+    */
+   if (!list_empty(&pc->lru))
+     list_move(&pc->lru, &pc_list);
+   else
+     continue;
+
+   if (__isolate_lru_page(page, mode) == 0) {
+     list_move(&page->lru, dst);
+     nr_taken++;
+   }
+ }
+
+ list_splice(&pc_list, src);
+ spin_unlock(&mem_cont->lru_lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
+@@ -151,6 +247,8 @@ int mem_cgroup_charge(struct page *pa

```

```

{
    struct mem_cgroup *mem;
    struct page_cgroup *pc, *race_pc;
+ unsigned long flags;
+ unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;

    /*
     * Should page_cgroup's go to their own slab?
@@ -197,7 +295,32 @@ int mem_cgroup_charge(struct page *pa
     * If we created the page_cgroup, we should free it on exceeding
     * the cgroup limit.
    */
- if (res_counter_charge(&mem->res, 1)) {
+ while (res_counter_charge(&mem->res, 1)) {
+ if (try_to_free_mem_cgroup_pages(mem))
+ continue;
+
+ /*
+  * try_to_free_mem_cgroup_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 cgroup.
+  * Check the limit again to see if the reclaim reduced the
+  * current usage of the cgroup before giving up
+  */
+ if (res_counter_check_under_limit(&mem->res))
+ continue;
+
+ /*
+  * Since we control both RSS and cache, we end up with a
+  * very interesting scenario where we end up reclaiming
+  * memory (essentially RSS), since the memory is pushed
+  * to swap cache, we eventually end up adding those
+  * pages back to our list. Hence we give ourselves a
+  * few chances before we fail
+  */
+ else if (nr_retries--) {
+ congestion_wait(WRITE, HZ/10);
+ continue;
+ }
+
    css_put(&mem->css);
    goto free_pc;
}
@@ -221,6 +344,10 @@ int mem_cgroup_charge(struct page *pa
    pc->page = page;
    page_assign_page_cgroup(page, pc);

+ spin_lock_irqsave(&mem->lru_lock, flags);
+ list_add(&pc->lru, &mem->active_list);

```

```

+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+
done:
    unlock_page_cgroup(page);
    return 0;
@@ -240,6 +367,7 @@ void mem_cgroup_uncharge(struct page_
{
    struct mem_cgroup *mem;
    struct page *page;
+ unsigned long flags;

    if (!pc)
        return;
@@ -252,6 +380,10 @@ void mem_cgroup_uncharge(struct page_
    page_assign_page_cgroup(page, NULL);
    unlock_page_cgroup(page);
    res_counter_uncharge(&mem->res, 1);
+
+ spin_lock_irqsave(&mem->lru_lock, flags);
+ list_del_init(&pc->lru);
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
    kfree(pc);
}
}
@@ -310,6 +442,7 @@ mem_cgroup_create(struct cgroup_su
    res_counter_init(&mem->res);
    INIT_LIST_HEAD(&mem->active_list);
    INIT_LIST_HEAD(&mem->inactive_list);
+ spin_lock_init(&mem->lru_lock);
    return &mem->css;
}

```

diff -puN mm/swap.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7 mm/swap.c

--- a/mm/swap.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7

+++ a/mm/swap.c

@@ -28,6 +28,7 @@

#include <linux/percpu.h>

#include <linux/cpu.h>

#include <linux/notifier.h>

+#include <linux/memcontrol.h>

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

int page_cluster;

@@ -145,6 +146,7 @@ void fastcall activate_page(struct page

SetPageActive(page);

add_page_to_active_list(zone, page);

__count_vm_event(PGACTIVATE);

+ mem_cgroup_move_lists(page_get_page_cgroup(page), true);

```

    }
    spin_unlock_irq(&zone->lru_lock);
}
diff -puN mm/vmscan.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7 mm/vmscan.c
--- a/mm/vmscan.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/mm/vmscan.c
@@ -37,6 +37,7 @@
#include <linux/delay.h>
#include <linux/kthread.h>
#include <linux/freezer.h>
+#include <linux/memcontrol.h>

#include <asm/tlbflush.h>
#include <asm/div64.h>
@@ -68,6 +69,15 @@ struct scan_control {
    int all_unreclaimable;

    int order;
+
+ /* Which cgroup do we reclaim from */
+ struct mem_cgroup *mem_cgroup;
+
+ /* 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_cgroup *mem_cont,
+ int active);
};

#define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -626,7 +636,7 @@ 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;

@@ -760,6 +770,21 @@ 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_cgroup *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.
@@ -801,11 +826,11 @@ 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_cgroup, 0);
nr_active = clear_active_flags(&page_list);
__count_vm_events(PGDEACTIVATE, nr_active);

@@ -1018,8 +1043,9 @@ 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_cgroup, 1);
zone->pages_scanned += pgscanned;
__mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
spin_unlock_irq(&zone->lru_lock);
@@ -1054,6 +1080,7 @@ force_reclaim_mapped:
ClearPageActive(page);

list_move(&page->lru, &zone->inactive_list);
+ mem_cgroup_move_lists(page_get_page_cgroup(page), false);
pgmoved++;
if (!pagevec_add(&pvec, page)) {
__mod_zone_page_state(zone, NR_INACTIVE, pgmoved);

```

```

@@ -1082,6 +1109,7 @@ force_reclaim_mapped:
    SetPageLRU(page);
    VM_BUG_ON(!PageActive(page));
    list_move(&page->lru, &zone->active_list);
+ mem_cgroup_move_lists(page_get_page_cgroup(page), true);
    pgmoved++;
    if (!pagevec_add(&pvec, page)) {
        __mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
@@ -1213,7 +1241,8 @@ 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;
@@ -1222,14 +1251,6 @@ 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,
- };

    count_vm_event(ALLOCSTALL);

@@ -1244,17 +1265,22 @@ 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 cgroups
+  */

```

```

+ if (sc->mem_cgroup == 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;
}
@@ -1266,18 +1292,18 @@ 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_cgroup == NULL)
  ret = 1;
out:
/*
@@ -1300,6 +1326,54 @@ 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_cgroup = NULL,
+ .isolate_pages = isolate_pages_global,
+ };
+
+ return do_try_to_free_pages(zones, gfp_mask, &sc);
+}
+
+#ifdef CONFIG_CGROUP_MEM_CONT
+
+#ifdef CONFIG_HIGHMEM
+#define ZONE_USERPAGES ZONE_HIGHMEM
+#else
+#define ZONE_USERPAGES ZONE_NORMAL
+#endif
+
+unsigned long try_to_free_mem_cgroup_pages(struct mem_cgroup *mem_cont)
+{
+ struct scan_control sc = {
+ .gfp_mask = GFP_KERNEL,
+ .may_writepage = !laptop_mode,
+ .may_swap = 1,
+ .swap_cluster_max = SWAP_CLUSTER_MAX,
+ .swappiness = vm_swappiness,
+ .order = 1,
+ .mem_cgroup = mem_cont,
+ .isolate_pages = mem_cgroup_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.
+ */
@@ -1335,6 +1409,8 @@ static unsigned long balance_pgdat(pg_da
+ .swap_cluster_max = SWAP_CLUSTER_MAX,
+ .swappiness = vm_swappiness,
+ .order = order,

```

```
+ .mem_cgroup = NULL,  
+ .isolate_pages = isolate_pages_global,  
};  
/*  
 * temp_priority is used to remember the scanning priority at which
```

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