

---

Subject: [PATCH O/4] Block I/O tracking

Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 09:22:51 GMT

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

---

Hi,

When you want to implement some kind of Block I/O controllers, you have to determine who issued each I/O. I just implemented this feature, with which you can track down the I/Os.

When you have to find the owner which issued the I/O, it is the one which owns the page where the IO is going to start. The cgroup memory subsystem already has this feature, so I realized that it would make easy to implemented Block I/O tracking mechanism on the memory subsystem. I named it "bio cgroup."

I made dm-ioband -- I/O bandwidth controller -- work with the bio cgroup, whose implementation is just experimental though.

I have a plan on making the bio cgroup support io\_context. Each bio cgroup will have one or more io\_contexts so the I/O bandwidth controller can use it to control the bandwidths.

I also have another plan on move the implementation of dm-ioband from the device mapper layer to somewhere before the I/O schedulers in the block layer.

The following patches are against linux-2.6.25-rc5-mm1 and you have to apply the patch of dm-ioband v0.0.3, which you can download from <http://people.valinux.co.jp/~ryov/dm-ioband/patches/dm-ioband-0.0.3.patch> before applying the following patches.

Let's say you want make two bio cgroups and assign them to ioband device "ioband1". First, you have to mount the bio cgroup filesystem.

```
# mount -t cgroup -o bio none /cgroup/bio
```

Then, you make new bio cgroups and put some processes in them.

```
# mkdir /cgroup/bio/bgroup1
# mkdir /cgroup/bio/bgroup2
# echo 1234 /cgroup/bio/bgroup1/tasks
# echo 5678 /cgroup/bio/bgroup1/tasks
```

Now you check the ids of the bio cgroups which you just created.

```
# cat /cgroup/bio/bgroup1/bio.id
1
# cat /cgroup/bio/bgroup2/bio.id
```

Finally, you can attach the cgroups to "ioband1" and assign them weights.

```
# dmsetup message ioband1 0 type cgroup
# dmsetup message ioband1 0 attach 1
# dmsetup message ioband1 0 attach 2
# dmsetup message ioband1 0 weight 1:30
# dmsetup message ioband1 0 weight 2:60
```

You can find the manual of dm-ioband at <http://people.valinux.co.jp/~ryov/dm-ioband/manual/index.html>. But the user interface for the bio cgroup is temporal and it will be changed after the io\_context support.

Thank you,  
Hirokazu Takahashi.

---

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

---



---

Subject: [PATCH 1/4] Block I/O tracking  
Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 09:25:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

This patch splits the cgroup memory subsystem into two parts. One is for tracking which cgroup which pages and the other is for controlling how much amount of memory should be assigned to each cgroup.

With this patch, you can use the page tracking mechanism even if the memory subsystem is off.

Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```
--- linux-2.6.25-rc5.pagecgroup/include/linux/memcontrol.h 2008-03-17 21:45:16.000000000
+0900
+++ linux-2.6.25-rc5-mm1/include/linux/memcontrol.h 2008-03-18 11:58:13.000000000 +0900
@@ -20,12 +20,61 @@
#ifdef _LINUX_MEMCONTROL_H
#define _LINUX_MEMCONTROL_H
```

```

+#include <linux/rcupdate.h>
+#include <linux/mm.h>
+#include <linux/smp.h>
+#include <linux/bit_spinlock.h>
+
+struct mem_cgroup;
+struct page_cgroup;
+struct page;
+struct mm_struct;

+#ifdef CONFIG_CGROUP_PAGE
+/*
+ * We use the lower bit of the page->page_cgroup pointer as a bit spin
+ * lock. We need to ensure that page->page_cgroup is at least two
+ * byte aligned (based on comments from Nick Piggin). But since
+ * bit_spin_lock doesn't actually set that lock bit in a non-debug
+ * uniprocessor kernel, we should avoid setting it here too.
+ */
+#define PAGE_CGROUP_LOCK_BIT 0x0
+#if defined(CONFIG_SMP) || defined(CONFIG_DEBUG_SPINLOCK)
+#define PAGE_CGROUP_LOCK (1 << PAGE_CGROUP_LOCK_BIT)
+#else
+#define PAGE_CGROUP_LOCK 0x0
+#endif
+
+/*
+ * A page_cgroup page is associated with every page descriptor. The
+ * page_cgroup helps us identify information about the cgroup
+ */
+struct page_cgroup {
+    #ifdef CONFIG_CGROUP_MEM_RES_CTLR
+    + struct list_head lru; /* per cgroup LRU list */
+    + struct mem_cgroup *mem_cgroup;
+    #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+    + struct page *page;
+    + int ref_cnt; /* cached, mapped, migrating */
+    + int flags;
+};
+#define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
+#define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
+
+static inline void lock_page_cgroup(struct page *page)
+{
+    + bit_spin_lock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+static inline int try_lock_page_cgroup(struct page *page)
+{

```

```

+ return bit_spin_trylock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+static inline void unlock_page_cgroup(struct page *page)
+{
+ bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}

extern void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p);
extern void mm_free_cgroup(struct mm_struct *mm);
@@ -38,41 +87,11 @@@ extern int mem_cgroup_charge(struct page
extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
    gfp_t gfp_mask);
extern void mem_cgroup_uncharge_page(struct page *page);
-extern void mem_cgroup_move_lists(struct page *page, 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);
-extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem, gfp_t gfp_mask);
-int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem);
-
-#define mm_match_cgroup(mm, cgroup) \
- ((cgroup) == rcu_dereference((mm)->mem_cgroup))
-
extern int mem_cgroup_prepare_migration(struct page *page);
extern void mem_cgroup_end_migration(struct page *page);
extern void mem_cgroup_page_migration(struct page *page, struct page *newpage);

-/*
- * For memory reclaim.
- */
-extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
-extern long mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);
-
-extern int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem);
-extern void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem,
- int priority);
-extern void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
- int priority);
-
-extern long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
- struct zone *zone, int priority);
-extern long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
- struct zone *zone, int priority);
-

```

```

-#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+#else /* CONFIG_CGROUP_PAGE */
static inline void mm_init_cgroup(struct mm_struct *mm,
    struct task_struct *p)
{
@@ -107,33 +126,69 @@ static inline void mem_cgroup_uncharge_p
{
}

-static inline void mem_cgroup_move_lists(struct page *page, bool active)
+static inline int mem_cgroup_prepare_migration(struct page *page)
{
+ return 0;
}

-static inline int mm_match_cgroup(struct mm_struct *mm, struct mem_cgroup *mem)
+static inline void mem_cgroup_end_migration(struct page *page)
{
- return 1;
}

-static inline int task_in_mem_cgroup(struct task_struct *task,
-    const struct mem_cgroup *mem)
+static inline void
+mem_cgroup_page_migration(struct page *page, struct page *newpage)
{
- return 1;
}
+#endif /* CONFIG_CGROUP_PAGE */

-static inline int mem_cgroup_prepare_migration(struct page *page)
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+
+extern void mem_cgroup_move_lists(struct page *page, 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);
+extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem, gfp_t gfp_mask);
+int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem);
+
+#define mm_match_cgroup(mm, cgroup) \
+ ((cgroup) == rcu_dereference((mm)->mem_cgroup))
+
+/*

```

```

+ * For memory reclaim.
+ */
+extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
+extern long mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);
+
+extern int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem);
+extern void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem,
+      int priority);
+extern void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
+      int priority);
+
+extern long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
+      struct zone *zone, int priority);
+extern long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+      struct zone *zone, int priority);
+
+#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+static inline void mem_cgroup_move_lists(struct page *page, bool active)
+{
+ return 0;
+}

-static inline void mem_cgroup_end_migration(struct page *page)
+static inline int mm_match_cgroup(struct mm_struct *mm, struct mem_cgroup *mem)
+{
+ return 1;
+}

-static inline void
-mem_cgroup_page_migration(struct page *page, struct page *newpage)
+static inline int task_in_mem_cgroup(struct task_struct *task,
+      const struct mem_cgroup *mem)
+{
+ return 1;
+}

static inline int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
--- linux-2.6.25-rc5.pagecgroup/include/linux/mm_types.h 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/include/linux/mm_types.h 2008-03-18 11:53:35.000000000 +0900
@@ -88,7 +88,7 @@ struct page {
void *virtual; /* Kernel virtual address (NULL if
not kmapped, ie. highmem) */
#endif /* WANT_PAGE_VIRTUAL */
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_CGROUP_PAGE
unsigned long page_cgroup;
#endif

```

```

#ifdef CONFIG_PAGE_OWNER
--- linux-2.6.25-rc5.pagecgroup/init/Kconfig 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/init/Kconfig 2008-03-18 11:53:35.000000000 +0900
@@ -379,6 +379,10 @@ config CGROUP_MEM_RES_CTLR
    Only enable when you're ok with these trade offs and really
    sure you need the memory resource controller.

+config CGROUP_PAGE
+    def_bool y
+    depends on CGROUP_MEM_RES_CTLR
+
config SYSFS_DEPRECATED
    bool

--- linux-2.6.25-rc5.pagecgroup/mm/Makefile 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/Makefile 2008-03-18 11:53:35.000000000 +0900
@@ -32,5 +32,5 @@ obj-$(CONFIG_FS_XIP) += filemap_xip.o
obj-$(CONFIG_MIGRATION) += migrate.o
obj-$(CONFIG_SMP) += allocpercpu.o
obj-$(CONFIG_QUICKLIST) += quicklist.o
-obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o
+obj-$(CONFIG_CGROUP_PAGE) += memcontrol.o

--- linux-2.6.25-rc5.pagecgroup/mm/memcontrol.c 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/memcontrol.c 2008-03-18 12:05:25.000000000 +0900
@@ -33,9 +33,15 @@

#include <asm/uaccess.h>

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
struct cgroup_subsys mem_cgroup_subsys;
static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+static inline int mem_cgroup_disabled(void)
+{
+ return mem_cgroup_subsys.disabled;
+}
+
/*
 * Statistics for memory cgroup.
 */
@@ -139,34 +145,6 @@ struct mem_cgroup {
};
static struct mem_cgroup init_mem_cgroup;

-/*
- * We use the lower bit of the page->page_cgroup pointer as a bit spin
- * lock. We need to ensure that page->page_cgroup is at least two

```

```

- * byte aligned (based on comments from Nick Piggin). But since
- * bit_spin_lock doesn't actually set that lock bit in a non-debug
- * uniprocessor kernel, we should avoid setting it here too.
- */
-#define PAGE_CGROUP_LOCK_BIT 0x0
-#if defined(CONFIG_SMP) || defined(CONFIG_DEBUG_SPINLOCK)
-#define PAGE_CGROUP_LOCK (1 << PAGE_CGROUP_LOCK_BIT)
-#else
-#define PAGE_CGROUP_LOCK 0x0
-#endif
-
-/*
- * A page_cgroup page is associated with every page descriptor. The
- * page_cgroup helps us identify information about the cgroup
- */
-struct page_cgroup {
- struct list_head lru; /* per cgroup LRU list */
- struct page *page;
- struct mem_cgroup *mem_cgroup;
- int ref_cnt; /* cached, mapped, migrating */
- int flags;
-};
-#define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
-#define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
-
- static int page_cgroup_nid(struct page_cgroup *pc)
- {
- return page_to_nid(pc->page);
@@ -177,11 +155,6 @@ static enum zone_type page_cgroup_zid(st
- return page_zonenum(pc->page);
- }

-enum charge_type {
- MEM_CGROUP_CHARGE_TYPE_CACHE = 0,
- MEM_CGROUP_CHARGE_TYPE_MAPPED,
-};
-
-/*
- * Always modified under lru lock. Then, not necessary to preempt_disable()
- */
@@ -242,51 +215,6 @@ static struct mem_cgroup *mem_cgroup_fro
- struct mem_cgroup, css);
- }

-void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
- {
- struct mem_cgroup *mem;
-

```



```

- mem = mem_cgroup_from_task(p);
- css_get(&mem->css);
- mm->mem_cgroup = mem;
-}
-
-void mm_free_cgroup(struct mm_struct *mm)
-{
- css_put(&mm->mem_cgroup->css);
-}
-
-
-static inline int page_cgroup_locked(struct page *page)
-{
- return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-
-static void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
-{
- VM_BUG_ON(!page_cgroup_locked(page));
- page->page_cgroup = ((unsigned long)pc | PAGE_CGROUP_LOCK);
-}
-
-
-struct page_cgroup *page_get_page_cgroup(struct page *page)
-{
- return (struct page_cgroup *) (page->page_cgroup & ~PAGE_CGROUP_LOCK);
-}
-
-
-static void lock_page_cgroup(struct page *page)
-{
- bit_spin_lock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-
-static int try_lock_page_cgroup(struct page *page)
-{
- return bit_spin_trylock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-
-static void unlock_page_cgroup(struct page *page)
-{
- bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-
static void __mem_cgroup_remove_list(struct page_cgroup *pc)
{
int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
@@ -519,253 +447,6 @@ unsigned long mem_cgroup_isolate_pages(u
}
}
/*

```

```

- * Charge the memory controller for page usage.
- * Return
- * 0 if the charge was successful
- * < 0 if the cgroup is over its limit
- */
-static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
-   gfp_t gfp_mask, enum charge_type ctype)
-{
- struct mem_cgroup *mem;
- struct page_cgroup *pc;
- unsigned long flags;
- unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
- struct mem_cgroup_per_zone *mz;
-
- if (mem_cgroup_subsys.disabled)
- return 0;
-
- /*
- * Should page_cgroup's go to their own slab?
- * One could optimize the performance of the charging routine
- * by saving a bit in the page_flags and using it as a lock
- * to see if the cgroup page already has a page_cgroup associated
- * with it
- */
-retry:
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- /*
- * The page_cgroup exists and
- * the page has already been accounted.
- */
- if (pc) {
- VM_BUG_ON(pc->page != page);
- VM_BUG_ON(pc->ref_cnt <= 0);
-
- pc->ref_cnt++;
- unlock_page_cgroup(page);
- goto done;
- }
- unlock_page_cgroup(page);
-
- pc = kzalloc(sizeof(struct page_cgroup), gfp_mask);
- if (pc == NULL)
- goto err;
-
- /*
- * We always charge the cgroup the mm_struct belongs to.
- * The mm_struct's mem_cgroup changes on task migration if the

```

```

- * thread group leader migrates. It's possible that mm is not
- * set, if so charge the init_mm (happens for pagecache usage).
- */
- if (!mm)
- mm = &init_mm;
-
- rcu_read_lock();
- mem = rcu_dereference(mm->mem_cgroup);
- /*
- * For every charge from the cgroup, increment reference count
- */
- css_get(&mem->css);
- rcu_read_unlock();
-
- while (res_counter_charge(&mem->res, PAGE_SIZE)) {
- if (!(gfp_mask & __GFP_WAIT))
- goto out;
-
- if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
- 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;
-
- if (!nr_retries--) {
- mem_cgroup_out_of_memory(mem, gfp_mask);
- goto out;
- }
- congestion_wait(WRITE, HZ/10);
- }
-
- pc->ref_cnt = 1;
- pc->mem_cgroup = mem;
- pc->page = page;
- pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
- if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
- pc->flags |= PAGE_CGROUP_FLAG_CACHE;
-
- lock_page_cgroup(page);
- if (page_get_page_cgroup(page)) {
- unlock_page_cgroup(page);

```

```

- /*
- * Another charge has been added to this page already.
- * We take lock_page_cgroup(page) again and read
- * page->cgroup, increment refcnt.... just retry is OK.
- */
- res_counter_uncharge(&mem->res, PAGE_SIZE);
- css_put(&mem->css);
- kfree(pc);
- goto retry;
- }
- page_assign_page_cgroup(page, pc);
-
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_add_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- unlock_page_cgroup(page);
-done:
- return 0;
-out:
- css_put(&mem->css);
- kfree(pc);
-err:
- return -ENOMEM;
-}
-
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm, gfp_t gfp_mask)
-{
- return mem_cgroup_charge_common(page, mm, gfp_mask,
- MEM_CGROUP_CHARGE_TYPE_MAPPED);
-}
-
-int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
- gfp_t gfp_mask)
-{
- if (!mm)
- mm = &init_mm;
- return mem_cgroup_charge_common(page, mm, gfp_mask,
- MEM_CGROUP_CHARGE_TYPE_CACHE);
-}
-
-/*
- * Uncharging is always a welcome operation, we never complain, simply
- * uncharge.
- */
-void mem_cgroup_uncharge_page(struct page *page)
-{

```

```

- struct page_cgroup *pc;
- struct mem_cgroup *mem;
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;
-
- if (mem_cgroup_subsys.disabled)
- return;
-
- /*
- * Check if our page_cgroup is valid
- */
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (!pc)
- goto unlock;
-
- VM_BUG_ON(pc->page != page);
- VM_BUG_ON(pc->ref_cnt <= 0);
-
- if (--(pc->ref_cnt) == 0) {
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_remove_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- page_assign_page_cgroup(page, NULL);
- unlock_page_cgroup(page);
-
- mem = pc->mem_cgroup;
- res_counter_uncharge(&mem->res, PAGE_SIZE);
- css_put(&mem->css);
-
- kfree(pc);
- return;
- }
-
-unlock:
- unlock_page_cgroup(page);
-}
-
-/*
- * Returns non-zero if a page (under migration) has valid page_cgroup member.
- * Refcnt of page_cgroup is incremented.
- */
-int mem_cgroup_prepare_migration(struct page *page)
- {
- struct page_cgroup *pc;
-
-

```

```

- if (mem_cgroup_subsys.disabled)
- return 0;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (pc)
- pc->ref_cnt++;
- unlock_page_cgroup(page);
- return pc != NULL;
-}
-
-void mem_cgroup_end_migration(struct page *page)
-{
- mem_cgroup_uncharge_page(page);
-}
-
-/*
- * We know both *page* and *newpage* are now not-on-LRU and PG_locked.
- * And no race with uncharge() routines because page_cgroup for *page*
- * has extra one reference by mem_cgroup_prepare_migration.
- */
-void mem_cgroup_page_migration(struct page *page, struct page *newpage)
-{
- struct page_cgroup *pc;
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (!pc) {
- unlock_page_cgroup(page);
- return;
- }
-
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_remove_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- page_assign_page_cgroup(page, NULL);
- unlock_page_cgroup(page);
-
- pc->page = newpage;
- lock_page_cgroup(newpage);
- page_assign_page_cgroup(newpage, pc);
-
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);

```

```

- __mem_cgroup_add_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- unlock_page_cgroup(newpage);
-}
-
-/*
 * This routine traverse page_cgroup in given list and drop them all.
 * This routine ignores page_cgroup->ref_cnt.
 * *And* this routine doesn't reclaim page itself, just removes page_cgroup.
@@ -812,7 +493,7 @@ static int mem_cgroup_force_empty(struct
int ret = -EBUSY;
int node, zid;

- if (mem_cgroup_subsys.disabled)
+ if (mem_cgroup_disabled())
    return 0;

    css_get(&mem->css);
@@ -1034,7 +715,7 @@ static void mem_cgroup_destroy(struct cg
static int mem_cgroup_populate(struct cgroup_subsys *ss,
    struct cgroup *cont)
{
- if (mem_cgroup_subsys.disabled)
+ if (mem_cgroup_disabled())
    return 0;
    return cgroup_add_files(cont, ss, mem_cgroup_files,
        ARRAY_SIZE(mem_cgroup_files));
@@ -1048,7 +729,7 @@ static void mem_cgroup_move_task(struct
struct mm_struct *mm;
struct mem_cgroup *mem, *old_mem;

- if (mem_cgroup_subsys.disabled)
+ if (mem_cgroup_disabled())
    return;

    mm = get_task_mm(p);
@@ -1086,3 +767,319 @@ struct cgroup_subsys mem_cgroup_subsys =
    .attach = mem_cgroup_move_task,
    .early_init = 0,
};
+#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+static inline int mem_cgroup_disabled(void)
+{
+ return 1;
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */

```

```

+
+void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
+{
+ struct mem_cgroup *mem;
+
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mem = mem_cgroup_from_task(p);
+ css_get(&mem->css);
+ mm->mem_cgroup = mem;
+ #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+}
+
+void mm_free_cgroup(struct mm_struct *mm)
+{
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ css_put(&mm->mem_cgroup->css);
+ #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+}
+
+static inline int page_cgroup_locked(struct page *page)
+{
+ return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+static void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
+{
+ VM_BUG_ON(!page_cgroup_locked(page));
+ page->page_cgroup = ((unsigned long)pc | PAGE_CGROUP_LOCK);
+}
+
+struct page_cgroup *page_get_page_cgroup(struct page *page)
+{
+ return (struct page_cgroup *) (page->page_cgroup & ~PAGE_CGROUP_LOCK);
+}
+
+enum charge_type {
+ MEM_CGROUP_CHARGE_TYPE_CACHE = 0,
+ MEM_CGROUP_CHARGE_TYPE_MAPPED,
+};
+
+ /*
+ * Charge the memory controller for page usage.
+ * Return
+ * 0 if the charge was successful
+ * < 0 if the cgroup is over its limit
+ */
+static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+ gfp_t gfp_mask, enum charge_type ctype)

```



```

+{
+ struct page_cgroup *pc;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct mem_cgroup *mem;
+ unsigned long flags;
+ unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+ struct mem_cgroup_per_zone *mz;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ if (mem_cgroup_disabled())
+ return 0;
+
+ /*
+ * Should page_cgroup's go to their own slab?
+ * One could optimize the performance of the charging routine
+ * by saving a bit in the page_flags and using it as a lock
+ * to see if the cgroup page already has a page_cgroup associated
+ * with it
+ */
+retry:
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ /*
+ * The page_cgroup exists and
+ * the page has already been accounted.
+ */
+ if (pc) {
+ VM_BUG_ON(pc->page != page);
+ VM_BUG_ON(pc->ref_cnt <= 0);
+
+ pc->ref_cnt++;
+ unlock_page_cgroup(page);
+ goto done;
+ }
+ unlock_page_cgroup(page);
+
+ pc = kzalloc(sizeof(struct page_cgroup), gfp_mask);
+ if (pc == NULL)
+ goto err;
+
+ /*
+ * We always charge the cgroup the mm_struct belongs to.
+ * The mm_struct's mem_cgroup changes on task migration if the
+ * thread group leader migrates. It's possible that mm is not
+ * set, if so charge the init_mm (happens for pagecache usage).
+ */
+ if (!mm)
+ mm = &init_mm;

```

```

+
+ rcu_read_lock();
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mem = rcu_dereference(mm->mem_cgroup);
+ /*
+  * For every charge from the cgroup, increment reference count
+  */
+ css_get(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ rcu_read_unlock();
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ while (res_counter_charge(&mem->res, PAGE_SIZE)) {
+ if (!(gfp_mask & __GFP_WAIT))
+ goto out;
+
+ if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
+ 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;
+
+ if (!nr_retries--) {
+ mem_cgroup_out_of_memory(mem, gfp_mask);
+ goto out;
+ }
+ congestion_wait(WRITE, HZ/10);
+ }
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ pc->ref_cnt = 1;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ pc->mem_cgroup = mem;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ pc->page = page;
+ pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
+ if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
+ pc->flags |= PAGE_CGROUP_FLAG_CACHE;
+
+ lock_page_cgroup(page);
+ if (page_get_page_cgroup(page)) {

```

```

+ unlock_page_cgroup(page);
+ /*
+  * Another charge has been added to this page already.
+  * We take lock_page_cgroup(page) again and read
+  * page->cgroup, increment refcnt... just retry is OK.
+  */
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ res_counter_uncharge(&mem->res, PAGE_SIZE);
+ css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ kfree(pc);
+ goto retry;
+ }
+ page_assign_page_cgroup(page, pc);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_add_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ unlock_page_cgroup(page);
+done:
+ return 0;
+out:
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ kfree(pc);
+err:
+ return -ENOMEM;
+}
+
+int mem_cgroup_charge(struct page *page, struct mm_struct *mm, gfp_t gfp_mask)
+{
+ return mem_cgroup_charge_common(page, mm, gfp_mask,
+ MEM_CGROUP_CHARGE_TYPE_MAPPED);
+}
+
+int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
+ gfp_t gfp_mask)
+{
+ if (!mm)
+ mm = &init_mm;
+ return mem_cgroup_charge_common(page, mm, gfp_mask,
+ MEM_CGROUP_CHARGE_TYPE_CACHE);
+}

```

```

+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_cgroup_uncharge_page(struct page *page)
+{
+ struct page_cgroup *pc;
+ struct mem_cgroup *mem;
+ struct mem_cgroup_per_zone *mz;
+ unsigned long flags;
+
+ if (mem_cgroup_disabled())
+ return;
+
+ /*
+ * Check if our page_cgroup is valid
+ */
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (!pc)
+ goto unlock;
+
+ VM_BUG_ON(pc->page != page);
+ VM_BUG_ON(pc->ref_cnt <= 0);
+
+ if (--(pc->ref_cnt) == 0) {
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_remove_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+ #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ page_assign_page_cgroup(page, NULL);
+ unlock_page_cgroup(page);
+
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mem = pc->mem_cgroup;
+ res_counter_uncharge(&mem->res, PAGE_SIZE);
+ css_put(&mem->css);
+ #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ kfree(pc);
+ return;
+ }
+
+unlock:

```

```

+ unlock_page_cgroup(page);
+}
+
+/*
+ * Returns non-zero if a page (under migration) has valid page_cgroup member.
+ * Refcnt of page_cgroup is incremented.
+ */
+int mem_cgroup_prepare_migration(struct page *page)
+{
+ struct page_cgroup *pc;
+
+ if (mem_cgroup_disabled())
+ return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc)
+ pc->ref_cnt++;
+ unlock_page_cgroup(page);
+ return pc != NULL;
+}
+
+void mem_cgroup_end_migration(struct page *page)
+{
+ mem_cgroup_uncharge_page(page);
+}
+
+/*
+ * We know both *page* and *newpage* are now not-on-LRU and PG_locked.
+ * And no race with uncharge() routines because page_cgroup for *page*
+ * has extra one reference by mem_cgroup_prepare_migration.
+ */
+void mem_cgroup_page_migration(struct page *page, struct page *newpage)
+{
+ struct page_cgroup *pc;
+ struct mem_cgroup_per_zone *mz;
+ unsigned long flags;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (!pc) {
+ unlock_page_cgroup(page);
+ return;
+ }
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);

```

```

+ __mem_cgroup_remove_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ page_assign_page_cgroup(page, NULL);
+ unlock_page_cgroup(page);
+
+ pc->page = newpage;
+ lock_page_cgroup(newpage);
+ page_assign_page_cgroup(newpage, pc);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_add_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ unlock_page_cgroup(newpage);
+}
+

```

---

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

---



---

Subject: [PATCH 2/4] Block I/O tracking  
Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 09:29:06 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

This patch implements the bio cgroup on the memory cgroup.

Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```

--- linux-2.6.25-rc5.pagecgroup2/include/linux/memcontrol.h 2008-03-18 12:45:14.000000000
+0900
+++ linux-2.6.25-rc5-mm1/include/linux/memcontrol.h 2008-03-18 12:55:59.000000000 +0900
@@ -54,6 +54,10 @@ struct page_cgroup {
    struct list_head lru; /* per cgroup LRU list */
    struct mem_cgroup *mem_cgroup;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+#ifdef CONFIG_CGROUP_BIO
+ struct list_head blist; /* for bio_cgroup page list */

```

```

+ struct bio_cgroup *bio_cgroup;
+#endif
  struct page *page;
  int ref_cnt; /* cached, mapped, migrating */
  int flags;
--- linux-2.6.25-rc5.pagecgroup2/include/linux/biocontrol.h 2008-03-17 22:06:49.000000000 +0900
+++ linux-2.6.25-rc5-mm1/include/linux/biocontrol.h 2008-03-18 14:19:53.000000000 +0900
@@ -0,0 +1,148 @@
+#include <linux/cgroup.h>
+#include <linux/mm.h>
+#include <linux/memcontrol.h>
+
+#ifndef _LINUX_BIOCONTROL_H
+#define _LINUX_BIOCONTROL_H
+
+#ifdef CONFIG_CGROUP_BIO
+
+struct io_context;
+
+struct bio_cgroup {
+ struct cgroup_subsys_state css;
+ int id;
+ struct io_context *io_context; /* default io_context */
+ /* struct radix_tree_root io_context_root; per device io_context */
+ spinlock_t page_list_lock;
+ struct list_head page_list;
+};
+
+static inline int bio_cgroup_disabled(void)
+{
+ return bio_cgroup_subsys.disabled;
+}
+
+extern void mm_init_bio_cgroup(struct mm_struct *, struct task_struct *);
+
+static inline void __bio_cgroup_add_page(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ list_add(&pc->blist, &biog->page_list);
+}
+
+static inline void bio_cgroup_add_page(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ unsigned long flags;
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ __bio_cgroup_add_page(pc);
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);

```

```

+}
+
+static inline void __bio_cgroup_remove_page(struct page_cgroup *pc)
+{
+ list_del_init(&pc->blist);
+}
+
+static inline void bio_cgroup_remove_page(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ unsigned long flags;
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ __bio_cgroup_remove_page(pc);
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);
+}
+
+static inline void get_bio_cgroup(struct bio_cgroup *biog)
+{
+ css_get(&biog->css);
+}
+
+static inline void put_bio_cgroup(struct bio_cgroup *biog)
+{
+ css_put(&biog->css);
+}
+
+static inline void set_bio_cgroup(struct page_cgroup *pc,
+ struct bio_cgroup *biog)
+{
+ pc->bio_cgroup = biog;
+}
+
+static inline void clear_bio_cgroup(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ pc->bio_cgroup = NULL;
+ put_bio_cgroup(biog);
+}
+
+/* This should be called in an RCU-protected section. */
+static inline struct bio_cgroup *mm_get_bio_cgroup(struct mm_struct *mm)
+{
+ struct bio_cgroup *biog = rcu_dereference(mm->bio_cgroup);
+ get_bio_cgroup(biog);
+ return biog;
+}
+
+static inline void mm_free_bio_cgroup(struct mm_struct *mm)

```



```

+{
+ put_bio_cgroup(mm->bio_cgroup);
+}
+
+extern int get_bio_cgroup_id(struct page *page);
+
+/* CONFIG_CGROUP_BIO */
+
+struct bio_cgroup;
+
+static inline int bio_cgroup_disabled(void)
+{
+     return 1;
+}
+
+static inline void mm_init_bio_cgroup(struct mm_struct *mm, struct task_struct *p)
+{
+}
+
+static inline void bio_cgroup_add_page(struct page_cgroup *pc)
+{
+}
+
+static inline void bio_cgroup_remove_page(struct page_cgroup *pc)
+{
+}
+
+static inline void get_bio_cgroup(struct bio_cgroup *biog)
+{
+}
+
+static inline void put_bio_cgroup(struct bio_cgroup *biog)
+{
+}
+
+static inline void set_bio_cgroup(struct page_cgroup *pc,
+ struct bio_cgroup *biog)
+{
+}
+
+static inline void clear_bio_cgroup(struct page_cgroup *pc)
+{
+}
+
+static inline struct bio_cgroup *mm_get_bio_cgroup(struct mm_struct *mm)
+{
+ return NULL;
+}

```

```

+
+static inline void mm_free_bio_cgroup(struct mm_struct *mm)
+{
+}
+
+static inline int get_bio_cgroup_id(struct page *page)
+{
+ return 0;
+}
+#endif /* CONFIG_CGROUP_BIO */
+
+#endif /* _LINUX_BIOCONTROL_H */
--- linux-2.6.25-rc5.pagecgroup2/include/linux/cgroup_subsys.h 2008-03-18 12:45:14.000000000
+0900
+++ linux-2.6.25-rc5-mm1/include/linux/cgroup_subsys.h 2008-03-18 12:55:59.000000000 +0900
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_BIO
+SUBSYS(bio_cgroup)
+#endif
+
+/* */
--- linux-2.6.25-rc5.pagecgroup2/include/linux/mm_types.h 2008-03-18 12:45:14.000000000
+0900
+++ linux-2.6.25-rc5-mm1/include/linux/mm_types.h 2008-03-18 12:55:59.000000000 +0900
@@ -230,6 +230,9 @@ struct mm_struct {
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR
 struct mem_cgroup *mem_cgroup;
 #endif
+#ifdef CONFIG_CGROUP_BIO
+ struct bio_cgroup *bio_cgroup;
+#endif

#ifdef CONFIG_PROC_FS
 /* store ref to file /proc/<pid>/exe symlink points to */
--- linux-2.6.25-rc5.pagecgroup2/init/Kconfig 2008-03-18 12:45:14.000000000 +0900
+++ linux-2.6.25-rc5-mm1/init/Kconfig 2008-03-18 12:55:59.000000000 +0900
@@ -379,9 +379,15 @@ config CGROUP_MEM_RES_CTLR
 Only enable when you're ok with these trade offs and really
 sure you need the memory resource controller.

+config CGROUP_BIO
+ bool "Block I/O cgroup subsystem"
+ depends on CGROUPS
+ help

```

```

+ Provides a Resource Controller that manages Block I/O.
+
config CGROUP_PAGE
    def_bool y
- depends on CGROUP_MEM_RES_CTLR
+ depends on CGROUP_MEM_RES_CTLR || CGROUP_BIO

config SYSFS_DEPRECATED
    bool
--- linux-2.6.25-rc5.pagecgroup2/mm/Makefile 2008-03-18 12:45:14.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/Makefile 2008-03-18 12:55:59.000000000 +0900
@@ -33,4 +33,5 @@ obj-$(CONFIG_MIGRATION) += migrate.o
obj-$(CONFIG_SMP) += allocpercpu.o
obj-$(CONFIG_QUICKLIST) += quicklist.o
obj-$(CONFIG_CGROUP_PAGE) += memcontrol.o
+obj-$(CONFIG_CGROUP_BIO) += biocontrol.o

--- linux-2.6.25-rc5.pagecgroup2/mm/biocontrol.c 2008-03-17 22:06:49.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/biocontrol.c 2008-03-18 12:55:59.000000000 +0900
@@ -0,0 +1,229 @@
+/* biocontrol.c - Block I/O Controller
+ *
+ * Copyright VA Linux Systems Japan, 2008
+ * Author Hirokazu Takahashi <taka@valinux.co.jp>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/mm.h>
+#include <linux/smp.h>
+#include <linux/bit_spinlock.h>
+#include <linux/idr.h>
+#include <linux/err.h>
+#include <linux/biocontrol.h>
+
+/* return corresponding bio_cgroup object of a cgroup */
+static inline struct bio_cgroup *cgroup_bio(struct cgroup *cgrp)
+{

```

```

+ return container_of(cgroup_subsys_state(cgrp, bio_cgroup_subsys_id),
+   struct bio_cgroup, css);
+}
+
+static inline struct bio_cgroup *bio_cgroup_from_task(struct task_struct *p)
+{
+ return container_of(task_subsys_state(p, bio_cgroup_subsys_id),
+   struct bio_cgroup, css);
+}
+
+void mm_init_bio_cgroup(struct mm_struct *mm, struct task_struct *p)
+{
+ struct bio_cgroup *biog;
+
+ biog = bio_cgroup_from_task(p);
+ get_bio_cgroup(biog);
+ mm->bio_cgroup = biog;
+}
+
+static struct idr bio_cgroup_idr;
+static DEFINE_SPINLOCK(bio_cgroup_idr_lock);
+
+static struct cgroup_subsys_state *
+bio_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
+{
+ struct bio_cgroup *biog;
+ int error;
+
+ biog = kzalloc(sizeof(*biog), GFP_KERNEL);
+ if (!biog)
+   return ERR_PTR(-ENOMEM);
+ if (!cgrp->parent) {
+   init_mm.bio_cgroup = biog;
+   idr_init(&bio_cgroup_idr);
+   biog->id = 0;
+ } else {
+retry:
+   if (unlikely(!idr_pre_get(&bio_cgroup_idr, GFP_KERNEL))) {
+     error = -EAGAIN;
+     goto out;
+   }
+   spin_lock_irq(&bio_cgroup_idr_lock);
+   error = idr_get_new_above(&bio_cgroup_idr, (void *)biog, 1, &biog->id);
+   spin_unlock_irq(&bio_cgroup_idr_lock);
+   if (error == -EAGAIN)
+     goto retry;
+   else if (error)
+     goto out;

```

```

+ }
+
+ INIT_LIST_HEAD(&biog->page_list);
+ spin_lock_init(&biog->page_list_lock);
+
+ /* Bind the cgroup to bio_cgroup object we just created */
+ biog->css.cgroup = cgrp;
+
+ return &biog->css;
+out:
+ kfree(biog);
+ return ERR_PTR(error);
+}
+
+#define FORCE_UNCHARGE_BATCH (128)
+static void bio_cgroup_force_empty(struct bio_cgroup *biog)
+{
+ struct page_cgroup *pc;
+ struct page *page;
+ int count = FORCE_UNCHARGE_BATCH;
+ struct list_head *list = &biog->page_list;
+ unsigned long flags;
+
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ while (!list_empty(list)) {
+ pc = list_entry(list->prev, struct page_cgroup, blist);
+ page = pc->page;
+ get_page(page);
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);
+ mem_cgroup_uncharge_page(page);
+ put_page(page);
+ if (--count <= 0) {
+ count = FORCE_UNCHARGE_BATCH;
+ cond_resched();
+ }
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ }
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);
+ return;
+}
+
+static void bio_cgroup_pre_destroy(struct cgroup_subsys *ss, struct cgroup *cgrp)
+{
+ struct bio_cgroup *biog = cgroup_bio(cgrp);
+ bio_cgroup_force_empty(biog);
+}
+
+static void bio_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cgrp)

```

```

+{
+ struct bio_cgroup *biog = cgroup_bio(cgrp);
+
+     spin_lock_irq(&bio_cgroup_idr_lock);
+     idr_remove(&bio_cgroup_id, biog->id);
+     spin_unlock_irq(&bio_cgroup_idr_lock);
+
+ kfree(biog);
+}
+
+struct bio_cgroup *find_bio_cgroup(int id)
+{
+ struct bio_cgroup *biog;
+     spin_lock_irq(&bio_cgroup_idr_lock);
+ biog = (struct bio_cgroup *)
+     idr_find(&bio_cgroup_id, id);
+     spin_unlock_irq(&bio_cgroup_idr_lock);
+ get_bio_cgroup(biog);
+ return biog;
+}
+
+int get_bio_cgroup_id(struct page *page)
+{
+ struct page_cgroup *pc;
+ int id = 0;
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc)
+     id = pc->bio_cgroup->id;
+ unlock_page_cgroup(page);
+ return id;
+}
+EXPORT_SYMBOL(get_bio_cgroup_id);
+
+static u64 bio_id_read(struct cgroup *cgrp, struct cftype *cft)
+{
+ struct bio_cgroup *biog = cgroup_bio(cgrp);
+
+ return (u64) biog->id;
+}
+
+
+static struct cftype bio_files[] = {
+ {
+     .name = "id",
+     .read_u64 = bio_id_read,
+ },
+};
+};

```

```

+
+static int bio_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ if (bio_cgroup_disabled())
+ return 0;
+ return cgroup_add_files(cont, ss, bio_files, ARRAY_SIZE(bio_files));
+}
+
+static void bio_cgroup_move_task(struct cgroup_subsys *ss,
+ struct cgroup *cont,
+ struct cgroup *old_cont,
+ struct task_struct *p)
+{
+ struct mm_struct *mm;
+ struct bio_cgroup *biog, *old_biog;
+
+ if (bio_cgroup_disabled())
+ return;
+
+ mm = get_task_mm(p);
+ if (mm == NULL)
+ return;
+
+ biog = cgroup_bio(cont);
+ old_biog = cgroup_bio(old_cont);
+
+ if (biog == old_biog)
+ goto out;
+
+ /*
+ * Only thread group leaders are allowed to migrate, the mm_struct is
+ * in effect owned by the leader
+ */
+ if (p->tgid != p->pid)
+ goto out;
+
+ get_bio_cgroup(biog);
+ rcu_assign_pointer(mm->bio_cgroup, biog);
+ put_bio_cgroup(old_biog);
+
+out:
+ mmput(mm);
+ return;
+}
+
+struct cgroup_subsys bio_cgroup_subsys = {
+ .name = "bio",

```

```

+ .subsys_id    = bio_cgroup_subsys_id,
+ .create      = bio_cgroup_create,
+ .destroy     = bio_cgroup_destroy,
+ .pre_destroy = bio_cgroup_pre_destroy,
+// .can_attach = bio_cgroup_can_attach,
+ .populate    = bio_cgroup_populate,
+ .attach     = bio_cgroup_move_task,
+ .early_init = 0,
+};
--- linux-2.6.25-rc5.pagecgroup2/mm/memcontrol.c 2008-03-18 12:45:14.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/memcontrol.c 2008-03-18 14:19:04.000000000 +0900
@@ -20,6 +20,7 @@
#include <linux/res_counter.h>
#include <linux/memcontrol.h>
#include <linux/cgroup.h>
+#include <linux/biocontrol.h>
#include <linux/mm.h>
#include <linux/smp.h>
#include <linux/page-flags.h>
@@ -784,6 +785,7 @@ void mm_init_cgroup(struct mm_struct *mm
    css_get(&mem->css);
    mm->mem_cgroup = mem;
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mm_init_bio_cgroup(mm, p);
}

void mm_free_cgroup(struct mm_struct *mm)
@@ -791,6 +793,7 @@ void mm_free_cgroup(struct mm_struct *mm
#ifdef CONFIG_CGROUP_MEM_RES_CTLR
    css_put(&mm->mem_cgroup->css);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mm_free_bio_cgroup(mm);
}

static inline int page_cgroup_locked(struct page *page)
@@ -830,8 +833,9 @@ static int mem_cgroup_charge_common(stru
    unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
    struct mem_cgroup_per_zone *mz;
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
-
- if (mem_cgroup_disabled())
+ struct bio_cgroup *biog;
+
+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
    return 0;

/*
@@ -879,6 +883,7 @@ retry:

```



```

*/
css_get(&mem->css);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ biog = mm_get_bio_cgroup(mm);
rcu_read_unlock();

#ifdef CONFIG_CGROUP_MEM_RES_CTLR
@@ -911,6 +916,7 @@ retry:
#ifdef CONFIG_CGROUP_MEM_RES_CTLR
pc->mem_cgroup = mem;
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ set_bio_cgroup(pc, biog);
pc->page = page;
pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
@@ -928,6 +934,7 @@ retry:
res_counter_uncharge(&mem->res, PAGE_SIZE);
css_put(&mem->css);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ clear_bio_cgroup(pc);
kfree(pc);
goto retry;
}
@@ -939,6 +946,7 @@ retry:
__mem_cgroup_add_list(pc);
spin_unlock_irqrestore(&mz->lru_lock, flags);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ bio_cgroup_add_page(pc);

unlock_page_cgroup(page);
done:
@@ -947,6 +955,7 @@ out:
#ifdef CONFIG_CGROUP_MEM_RES_CTLR
css_put(&mem->css);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ put_bio_cgroup(biog);
kfree(pc);
err:
return -ENOMEM;
@@ -978,7 +987,7 @@ void mem_cgroup_uncharge_page(struct pag
struct mem_cgroup_per_zone *mz;
unsigned long flags;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
return;

/*

```

```

@@ -999,6 +1008,7 @@ void mem_cgroup_uncharge_page(struct pag
    __mem_cgroup_remove_list(pc);
    spin_unlock_irqrestore(&mz->lru_lock, flags);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ bio_cgroup_remove_page(pc);

    page_assign_page_cgroup(page, NULL);
    unlock_page_cgroup(page);
@@ -1008,6 +1018,7 @@ void mem_cgroup_uncharge_page(struct pag
    res_counter_uncharge(&mem->res, PAGE_SIZE);
    css_put(&mem->css);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ clear_bio_cgroup(pc);

    kfree(pc);
    return;
@@ -1025,7 +1036,7 @@ int mem_cgroup_prepare_migration(struct
{
    struct page_cgroup *pc;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
    return 0;

    lock_page_cgroup(page);
@@ -1065,6 +1076,7 @@ void mem_cgroup_page_migration(struct pa
    __mem_cgroup_remove_list(pc);
    spin_unlock_irqrestore(&mz->lru_lock, flags);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ bio_cgroup_remove_page(pc);

    page_assign_page_cgroup(page, NULL);
    unlock_page_cgroup(page);
@@ -1079,6 +1091,7 @@ void mem_cgroup_page_migration(struct pa
    __mem_cgroup_add_list(pc);
    spin_unlock_irqrestore(&mz->lru_lock, flags);
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ bio_cgroup_add_page(pc);

    unlock_page_cgroup(newpage);
}

```

---

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

---

Subject: [PATCH 3/4] Block I/O tracking

Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 09:30:33 GMT

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

---

Hi,

This patch is for cleaning up the code of the cgroup memory subsystem to remove some "#ifdef"s.

Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```
--- linux-2.6.25-rc5.pagecgroup3/mm/memcontrol.c 2008-03-18 13:01:57.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/memcontrol.c 2008-03-18 14:00:23.000000000 +0900
@@ -216,6 +216,51 @@ static struct mem_cgroup *mem_cgroup_fro
     struct mem_cgroup, css);
 }

+static inline void get_mem_cgroup(struct mem_cgroup *mem)
+{
+ css_get(&mem->css);
+}
+
+static inline void put_mem_cgroup(struct mem_cgroup *mem)
+{
+ css_put(&mem->css);
+}
+
+static inline void set_mem_cgroup(struct page_cgroup *pc,
+                                struct mem_cgroup *mem)
+{
+ pc->mem_cgroup = mem;
+}
+
+static inline void clear_mem_cgroup(struct page_cgroup *pc)
+{
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ res_counter_uncharge(&mem->res, PAGE_SIZE);
+ pc->mem_cgroup = NULL;
+ put_mem_cgroup(mem);
+}
+
+/* This should be called in an RCU-protected section. */
+static inline struct mem_cgroup *mm_get_mem_cgroup(struct mm_struct *mm)
+{
+ struct mem_cgroup *mem = rcu_dereference(mm->mem_cgroup);
+ get_mem_cgroup(mem);
+ return mem;
}
```

```

+}
+
+static inline void mm_init_mem_cgroup(struct mm_struct *mm,
+ struct task_struct *p)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_task(p);
+ get_mem_cgroup(mem);
+ mm->mem_cgroup = mem;
+}
+
+static inline void mm_free_mem_cgroup(struct mm_struct *mm)
+{
+ put_mem_cgroup(mm->mem_cgroup);
+}
+
+static void __mem_cgroup_remove_list(struct page_cgroup *pc)
+{
+ int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
@@ -266,6 +311,26 @@ static void __mem_cgroup_move_lists(stru
+}
+}

+static inline void mem_cgroup_add_page(struct page_cgroup *pc)
+{
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+ unsigned long flags;
+
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_add_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+}
+
+static inline void mem_cgroup_remove_page(struct page_cgroup *pc)
+{
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+ unsigned long flags;
+
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_remove_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+}
+
+int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
+{
+ int ret;
@@ -305,6 +370,37 @@ void mem_cgroup_move_lists(struct page *
+ unlock_page_cgroup(page);
+}

```

```

+static inline int mem_cgroup_try_to_allocate(struct mem_cgroup *mem,
+ gfp_t gfp_mask)
+{
+ unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+
+ while (res_counter_charge(&mem->res, PAGE_SIZE)) {
+ if (!(gfp_mask & __GFP_WAIT))
+ return -1;
+
+ if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
+ 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;
+
+ if (!nr_retries--) {
+ mem_cgroup_out_of_memory(mem, gfp_mask);
+ return -1;
+ }
+ congestion_wait(WRITE, HZ/10);
+ }
+ return 0;
+}
+
+/*
+ * Calculate mapped_ratio under memory controller. This will be used in
+ * vmscan.c for determining we have to reclaim mapped pages.
+@@ -497,7 +593,7 @@ static int mem_cgroup_force_empty(struct
+ if (mem_cgroup_disabled())
+ return 0;
+
+ - css_get(&mem->css);
+ + get_mem_cgroup(mem);
+ /*
+ * page reclaim code (kswapd etc..) will move pages between
+ * active_list <-> inactive_list while we don't take a lock.
+@@ -518,7 +614,7 @@ static int mem_cgroup_force_empty(struct
+ }
+ ret = 0;
+ out:

```

```

- css_put(&mem->css);
+ put_mem_cgroup(mem);
  return ret;
}

@@ -750,9 +846,9 @@ static void mem_cgroup_move_task(struct
  if (p->tgid != p->pid)
    goto out;

- css_get(&mem->css);
+ get_mem_cgroup(mem);
  rcu_assign_pointer(mm->mem_cgroup, mem);
- css_put(&old_mem->css);
+ put_mem_cgroup(old_mem);

out:
  mmput(mm);
@@ -770,29 +866,47 @@ struct cgroup_subsys mem_cgroup_subsys =
};
#else /* CONFIG_CGROUP_MEM_RES_CTLR */

+struct mem_cgroup;
+
static inline int mem_cgroup_disabled(void)
{
  return 1;
}
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */

-void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
+static inline void mem_cgroup_add_page(struct page_cgroup *pc) {}
+static inline void mem_cgroup_remove_page(struct page_cgroup *pc) {}
+static inline void get_mem_cgroup(struct mem_cgroup *mem) {}
+static inline void put_mem_cgroup(struct mem_cgroup *mem) {}
+static inline void set_mem_cgroup(struct page_cgroup *pc,
+  struct mem_cgroup *mem) {}
+static inline void clear_mem_cgroup(struct page_cgroup *pc) {}
+
+static inline struct mem_cgroup *mm_get_mem_cgroup(struct mm_struct *mm)
{
- struct mem_cgroup *mem;
+ return NULL;
+}

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mem = mem_cgroup_from_task(p);
- css_get(&mem->css);
- mm->mem_cgroup = mem;

```

```

+static inline void mm_init_mem_cgroup(struct mm_struct *mm,
+ struct task_struct *p) {}
+static inline void mm_free_mem_cgroup(struct mm_struct *mm) {}
+
+static inline int mem_cgroup_try_to_allocate(struct mem_cgroup *mem,
+ gfp_t gfp_mask)
+{
+ return 0;
+}
+
+ #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
+{
+ mm_init_mem_cgroup(mm, p);
+ mm_init_bio_cgroup(mm, p);
+}

void mm_free_cgroup(struct mm_struct *mm)
{
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- css_put(&mm->mem_cgroup->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mm_free_mem_cgroup(mm);
+ mm_free_bio_cgroup(mm);
+}

@@ -827,12 +941,7 @@ static int mem_cgroup_charge_common(stru
+ gfp_t gfp_mask, enum charge_type ctype)
+{
+ struct page_cgroup *pc;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct mem_cgroup *mem;
- unsigned long flags;
- unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
- struct mem_cgroup_per_zone *mz;
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ struct bio_cgroup *biog;

+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
@@ -876,46 +985,18 @@ retry:
+ mm = &init_mm;

+ rcu_read_lock();
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mem = rcu_dereference(mm->mem_cgroup);
+ /*
+ * For every charge from the cgroup, increment reference count

```

```

 */
- css_get(&mem->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mem = mm_get_mem_cgroup(mm);
  biog = mm_get_bio_cgroup(mm);
  rcu_read_unlock();

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- while (res_counter_charge(&mem->res, PAGE_SIZE)) {
- if (!(gfp_mask & __GFP_WAIT))
- goto out;
-
- if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
- 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;
-
- if (!nr_retries--) {
- mem_cgroup_out_of_memory(mem, gfp_mask);
- goto out;
- }
- congestion_wait(WRITE, HZ/10);
- }
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ if (mem_cgroup_try_to_allocate(mem, gfp_mask) < 0)
+ goto out;

  pc->ref_cnt = 1;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- pc->mem_cgroup = mem;
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ set_mem_cgroup(pc, mem);
  set_bio_cgroup(pc, biog);
  pc->page = page;
  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
@@ -930,31 +1011,21 @@ retry:
  * We take lock_page_cgroup(page) again and read
  * page->cgroup, increment refcnt.... just retry is OK.
  */
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR

```



```

- res_counter_uncharge(&mem->res, PAGE_SIZE);
- css_put(&mem->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ clear_mem_cgroup(pc);
  clear_bio_cgroup(pc);
  kfree(pc);
  goto retry;
}
page_assign_page_cgroup(page, pc);

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_add_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mem_cgroup_add_page(pc);
  bio_cgroup_add_page(pc);

  unlock_page_cgroup(page);
done:
  return 0;
out:
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- css_put(&mem->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ put_mem_cgroup(mem);
  put_bio_cgroup(biog);
  kfree(pc);
err:
@@ -983,9 +1054,6 @@ int mem_cgroup_cache_charge(struct page
void mem_cgroup_uncharge_page(struct page *page)
{
  struct page_cgroup *pc;
- struct mem_cgroup *mem;
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;

  if (mem_cgroup_disabled() && bio_cgroup_disabled())
    return;
@@ -1002,22 +1070,13 @@ void mem_cgroup_uncharge_page(struct pag
  VM_BUG_ON(pc->ref_cnt <= 0);

  if (--(pc->ref_cnt) == 0) {
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_remove_list(pc);

```

```

- spin_unlock_irqrestore(&mz->lru_lock, flags);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mem_cgroup_remove_page(pc);
  bio_cgroup_remove_page(pc);

  page_assign_page_cgroup(page, NULL);
  unlock_page_cgroup(page);

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mem = pc->mem_cgroup;
- res_counter_uncharge(&mem->res, PAGE_SIZE);
- css_put(&mem->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ clear_mem_cgroup(pc);
  clear_bio_cgroup(pc);

  kfree(pc);
@@ -1060,8 +1119,6 @@ void mem_cgroup_end_migration(struct pag
void mem_cgroup_page_migration(struct page *page, struct page *newpage)
{
  struct page_cgroup *pc;
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;

  lock_page_cgroup(page);
  pc = page_get_page_cgroup(page);
@@ -1070,12 +1127,7 @@ void mem_cgroup_page_migration(struct pa
return;
}

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_remove_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mem_cgroup_remove_page(pc);
  bio_cgroup_remove_page(pc);

  page_assign_page_cgroup(page, NULL);
@@ -1085,12 +1137,7 @@ void mem_cgroup_page_migration(struct pa
lock_page_cgroup(newpage);
page_assign_page_cgroup(newpage, pc);

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_add_list(pc);

```

```
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mem_cgroup_add_page(pc);
  bio_cgroup_add_page(pc);
```

```
unlock_page_cgroup(newpage);
```

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: [PATCH 4/4] Block I/O tracking

Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 09:31:48 GMT

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

---

Hi,

With this patch, dm-ioband can work with the bio cgroup.

Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```
--- linux-2.6.25-rc3.ioload/drivers/md/dm-ioband-type.c 2008-03-13 22:27:45.000000000 +0900
+++ linux-2.6.25-rc3-mm1/drivers/md/dm-ioband-type.c 2008-03-17 15:57:32.000000000 +0900
```

```
@ @ -6,6 +6,7 @ @
```

```
* This file is released under the GPL.
```

```
*/
```

```
#include <linux/bio.h>
```

```
+#include <linux/biocontrol.h>
```

```
#include "dm.h"
```

```
#include "dm-bio-list.h"
```

```
#include "dm-ioband.h"
```

```
@ @ -57,13 +58,7 @ @ static int ioband_node(struct bio *bio)
```

```
static int ioband_cgroup(struct bio *bio)
```

```
{
```

```
- /*
```

```
- * This function should return the ID of the cgroup which issued "bio".
```

```
- * The ID of the cgroup which the current process belongs to won't be
```

```
- * suitable ID for this purpose, since some BIOs will be handled by kernel
```

```
- * threads like aio or pdflush on behalf of the process requesting the BIOs.
```

```
- */
```

```
- return 0; /* not implemented yet */
```

```
+ return get_bio_cgroup_id(bio->bi_io_vec[0].bv_page);
```

```
}
```

```
struct group_type dm_ioband_group_type[] = {
```

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 0/4] Block I/O tracking

Posted by [KAMEZAWA Hiroyuki](#) on Tue, 18 Mar 2008 10:15:11 GMT

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

---

On Tue, 18 Mar 2008 18:22:51 +0900 (JST)

Hirokazu Takahashi <taka@valinux.co.jp> wrote:

> # mount -t cgroup -o bio none /cgroup/bio

>

> Then, you make new bio cgroups and put some processes in them.

>

> # mkdir /cgroup/bio/bgroup1

> # mkdir /cgroup/bio/bgroup2

> # echo 1234 /cgroup/bio/bgroup1/tasks

> # echo 5678 /cgroup/bio/bgroup1/tasks

>

> Now you check the ids of the bio cgroups which you just created.

>

> # cat /cgroup/bio/bgroup1/bio.id

> 1

> # cat /cgroup/bio/bgroup2/bio.id

> 2

>

> Finally, you can attach the cgroups to "ioband1" and assign them weights.

>

> # dmsetup message ioband1 0 type cgroup

> # dmsetup message ioband1 0 attach 1

> # dmsetup message ioband1 0 attach 2

> # dmsetup message ioband1 0 weight 1:30

> # dmsetup message ioband1 0 weight 2:60

>

> You can find the manual of dm-ioband at

> <http://people.valinux.co.jp/~ryov/dm-ioband/manual/index.html>.

> But the user interface for the bio cgroup is temporal and it will be

> changed after the io\_context support.

>

I'm glad if these some kinds of params rather than 'id' are also shown under cgroup.

Thanks,

-Kame

---

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

---

---

Subject: Re: [PATCH 2/4] Block I/O tracking  
Posted by [KAMEZAWA Hiroyuki](#) on Tue, 18 Mar 2008 10:20:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 18 Mar 2008 18:29:06 +0900 (JST)  
Hirokazu Takahashi <taka@valinux.co.jp> wrote:

```
> Hi,  
>  
> This patch implements the bio cgroup on the memory cgroup.  
>  
> Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>  
>  
>  
> --- linux-2.6.25-rc5.pagecgroup2/include/linux/memcontrol.h 2008-03-18 12:45:14.000000000  
+0900  
> +++ linux-2.6.25-rc5-mm1/include/linux/memcontrol.h 2008-03-18 12:55:59.000000000 +0900  
> @@ -54,6 +54,10 @@ struct page_cgroup {  
> struct list_head lru; /* per cgroup LRU list */  
> struct mem_cgroup *mem_cgroup;  
> #endif /* CONFIG_CGROUP_MEM_RES_CTLR */  
> +#ifdef CONFIG_CGROUP_BIO  
> + struct list_head blist; /* for bio_cgroup page list */  
> + struct bio_cgroup *bio_cgroup;  
> +#endif
```

Hmm, definition like this

```
==  
enum {  
#ifdef CONFIG_CGROUP_MEM_RES_CTLR  
MEM_RES_CTLR,  
#endif  
#ifdef CONFIG_CGROUP_BIO  
BIO_CTLR,  
#endif  
NR_VM_CTRL,  
};
```

```
void *cgroups[NR_VM_CGROUP];
```

```
==
```

Can save another #ifdefs ?

And, blist seems to be just used for force\_empty.

Do you really need this ? no alternative ?

Thanks,

-Kame

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 1/4] Block I/O tracking

Posted by [KAMEZAWA Hiroyuki](#) on Tue, 18 Mar 2008 10:24:02 GMT

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

---

On Tue, 18 Mar 2008 18:25:08 +0900 (JST)

Hirokazu Takahashi <taka@valinux.co.jp> wrote:

> Hi,

>

> This patch splits the cgroup memory subsystem into two parts.

> One is for tracking which cgroup which pages and the other is

> for controlling how much amount of memory should be assigned to

> each cgroup.

>

> With this patch, you can use the page tracking mechanism even if

> the memory subsystem is off.

>

> Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

>

I think current my work, radix-tree page cgroup will help this work.

It creates page\_cgroup.h and page\_cgroup.c.

Patches are posted last week to the mm-list.(And now rewriting..)

Please let me know if you have requests.

Thanks,

-Kame

---

Containers mailing list

Containers@lists.linux-foundation.org

---

---

Subject: Re: [PATCH 2/4] Block I/O tracking  
Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 11:34:22 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

```
> > Hi,
> >
> > This patch implements the bio cgroup on the memory cgroup.
> >
> > Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>
> >
> >
> > --- linux-2.6.25-rc5.pagecgroup2/include/linux/memcontrol.h 2008-03-18 12:45:14.000000000
+0900
> > +++ linux-2.6.25-rc5-mm1/include/linux/memcontrol.h 2008-03-18 12:55:59.000000000 +0900
> > @@ -54,6 +54,10 @@ struct page_cgroup {
> > struct list_head lru; /* per cgroup LRU list */
> > struct mem_cgroup *mem_cgroup;
> > #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
> > +#ifdef CONFIG_CGROUP_BIO
> > + struct list_head blist; /* for bio_cgroup page list */
> > + struct bio_cgroup *bio_cgroup;
> > +#endif
>
> Hmm, definition like this
> ==
> enum {
> #ifdef CONFIG_CGROUP_MEM_RES_CTLR
> MEM_RES_CTLR,
> #endif
> #ifdef CONFIG_CGROUP_BIO
> BIO_CTLR,
> #endif
> NR_VM_CTRL,
> };
>
> void *cgroups[NR_VM_CGROUP];
> ==
> Can save another #ifdefs ?
```

I guess the following code can be possible if you really want to remove the #ifdefs here. But the both of them look overkill.

```
#ifdef CONFIG_CGROUP_MEM_RES_CTLR
```

```
#define MEM_CONTROL \  
    struct list_head lru; \  
    struct mem_cgroup *mem_cgroup \  
#else  
#define MEM_CONTROL  
#endif  
  
#define BIO_CONTROL \  
    struct list_head blist; \  
    struct bio_cgroup *bio_cgroup \  
#else  
#define BIO_CONTROL  
#endif  
  
struct page_cgroup {  
    MEM_CONTROL;  
    BIO_CONTROL;  
    struct page *page;  
    int ref_cnt;  
    int flags;  
};
```

> And, blist seems to be just used for force\_empty.  
> Do you really need this ? no alternative ?

I selected this approach because it was the simplest way for the first implementation.

I've been also thinking about what you pointed.  
If you don't mind taking a long time to remove a bio cgroup, it will be the easiest way that you can scan all pages to find the pages which belong to the cgroup and delete them. It may be enough since you may say it will rarely happen. But it might cause some trouble on machines with huge memory.

So I'm actually looking for another way if we could release them lazily. But I don't think I have to hurry to work on this issue now.

> Thanks,  
> -Kame

Thank you,  
Hirokazu Takahashi.

---

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

---



Subject: Re: [PATCH 2/4] Block I/O tracking  
Posted by [KAMEZAWA Hiroyuki](#) on Tue, 18 Mar 2008 11:52:18 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 18 Mar 2008 20:34:22 +0900 (JST)  
Hirokazu Takahashi <taka@valinux.co.jp> wrote:

>  
> > And, blist seems to be just used for force\_empty.  
> > Do you really need this ? no alternative ?  
>  
> I selected this approach because it was the simplest way for the  
> first implementation.  
>  
> I've been also thinking about what you pointed.  
> If you don't mind taking a long time to remove a bio cgroup, it will be  
> the easiest way that you can scan all pages to find the pages which  
> belong to the cgroup and delete them. It may be enough since you may  
> say it will rarely happen. But it might cause some trouble on machines  
> with huge memory.  
>  
Hmm, force\_empty itself is necessary ?

Thanks,  
-Kame

---

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

---

---

Subject: Re: [PATCH 2/4] Block I/O tracking  
Posted by [Hirokazu Takahashi](#) on Tue, 18 Mar 2008 12:15:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

> > > And, blist seems to be just used for force\_empty.  
> > > Do you really need this ? no alternative ?  
> >  
> > I selected this approach because it was the simplest way for the  
> > first implementation.  
> >  
> > I've been also thinking about what you pointed.  
> > If you don't mind taking a long time to remove a bio cgroup, it will be  
> > the easiest way that you can scan all pages to find the pages which  
> > belong to the cgroup and delete them. It may be enough since you may

> > say it will rarely happen. But it might cause some trouble on machines  
> > with huge memory.  
> >  
> Hmm, force\_empty itself is necessary ?

It is called when bio cgroups are removed.  
With the current implementation, when you delete a bio cgroup,  
the bio\_cgroup members of page\_cgroups which point the cgroup  
have to be cleared.

So I'm looking for another way like:

- Use some kind of id instead of a pointer to a bio cgroup,  
so you can check whether the id is valid before you use it.
- Don't free the bio cgroup until all the pages referring to  
the cgroup.

I also want to implement that if you find a page whose cgroup is  
already removed, the page should be assigned to a new cgroup.

Thank you,  
Hirokazu Takahashi.

---

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

---

---

Subject: Re: [PATCH O/4] Block I/O tracking  
Posted by [Hirokazu Takahashi](#) on Wed, 19 Mar 2008 03:59:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

```
> > # mount -t cgroup -o bio none /cgroup/bio
> >
> > Then, you make new bio cgroups and put some processes in them.
> >
> > # mkdir /cgroup/bio/bgroup1
> > # mkdir /cgroup/bio/bgroup2
> > # echo 1234 /cgroup/bio/bgroup1/tasks
> > # echo 5678 /cgroup/bio/bgroup1/tasks
> >
> > Now you check the ids of the bio cgroups which you just created.
> >
> > # cat /cgroup/bio/bgroup1/bio.id
> > 1
> > # cat /cgroup/bio/bgroup2/bio.id
```

```
> > 2
> >
> > Finally, you can attach the cgroups to "ioband1" and assign them weights.
> >
> > # dmsetup message ioband1 0 type cgroup
> > # dmsetup message ioband1 0 attach 1
> > # dmsetup message ioband1 0 attach 2
> > # dmsetup message ioband1 0 weight 1:30
> > # dmsetup message ioband1 0 weight 2:60
> >
> > You can find the manual of dm-ioband at
> > http://people.valinux.co.jp/~ryov/dm-ioband/manual/index.html.
> > But the user interface for the bio cgroup is temporal and it will be
> > changed after the io_context support.
> >
> > I'm glad if these some kinds of params rather than 'id' are also shown
> > under cgroup.
```

You mean each bio cgroup has to have a lot of files which shows the status of the cgroup or allows you to control the cgroup. I think this should be done after the cgroup bio subsystem supports io\_context since the interface will be changed to support it.

```
> Thanks,
> -Kame
```

Thank you,  
Hirokazu Takahashi.

---

Containers mailing list  
[Containers@lists.linux-foundation.org](mailto:Containers@lists.linux-foundation.org)  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---