Subject: [PATCH O/4] BIO tracking take2
Posted by Hirokazu Takahashi on Tue, 20 May 2008 11:59:49 GMT
View Forum Message <> Reply to Message

Hi all,

With this series of patches, you can determine the owners of any
type of I/Os. I ported the previous version to linux-2.6.26-rc2-mm1.
This makes dm-ioband -- I/O bandwidth controller -- be able to control
the Block I/O bandwidths even when it accepts delayed write requests.
Dm-ioband can find the owner cgroup of each request.
It is also possible that OpenVz team and NEC Uchida-san team working on
the CFQ scheduler use this functionality to control asynchronous I/Os
with a little enhancement.

You have to apply the patch dm-ioband v1.0.0 before applying this series
of patches, which can be found at:
http://people.valinux.co.jp/~ryov/dm-ioband

And you have to select the following config options when compiling kernel:
  CONFIG_CGROUPS=y
  CONFIG_CGROUP_BIO=y
And I recommend you should also select the options for cgroup memory
subsystem, because it makes it possible to give some I/O bandwidth
and some memory to a certain cgroup to control delayed write requests
and the processes in the cgroup will be able to make pages dirty only
inside the cgroup even when the given bandwidth is narrow.
  CONFIG_RESOURCE_COUNTERS=y
  CONFIG_CGROUP_MEM_RES_CTLR=y

This code is based on some part of the memory subsystem of cgroup
and I think the accuracy and overhead of the subsystem can't be ignored
at this time, so we need to keep tuning it up.

The following shows how to use dm-ioband with cgroups.
Please assume that you want make two cgroups, which we call "bio cgroup"
here, to track down block I/Os and assign them to ioband device "ioband1".

First, mount the bio cgroup filesystem.

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

Then, 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, check the ID of each bio cgroup which is just created.

```
# cat /cgroup/bio/bgroup1/bio.id
  1
# cat /cgroup/bio/bgroup2/bio.id
  2
```

Finally, 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 also make use of the dm-ioband administration tool if you want.
The tool will be found here:
http://people.valinux.co.jp/~kaizuka/dm-ioband/iobandctl/manual.html
You can set up the device with the tool as follows.
In this case, you don't need to know the IDs of the cgroups.

```
# iobandctl.py group /dev/mapper/ioband1 cgroup /cgroup/bio/bgroup1:30 /cgroup/bio/bgroup2:60
```

Thank you,
Hirokazu Takahashi.

Subject: [PATCH 1/4] BIO tracking take2
Posted by Hirokazu Takahashi on Tue, 20 May 2008 12:02:00 GMT
View Forum Message <> Reply to Message

Hi,

This patch splits the cgroup memory subsystem into two parts.
One is for tracking pages to find out the owners. 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>

```
diff -udpr linux-2.6.26-rc2.cg0/include/linux/memcontrol.h
linux-2.6.26-rc2/include/linux/memcontrol.h
--- linux-2.6.26-rc2.cg0/include/linux/memcontrol.h 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/memcontrol.h 2008-05-16 19:49:51.000000000 +0900
@@ -20,12 +20,61 @@
 #ifndef _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);
+}

 #define page_reset_bad_cgroup(page) ((page)->page_cgroup = 0)

@@ -35,44 +84,15 @@ 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);
-
-extern struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p);
-
-#define mm_match_cgroup(mm, cgroup) \
- ((cgroup) == mem_cgroup_from_task((mm)->owner))

 extern int
 mem_cgroup_prepare_migration(struct page *page, struct page *newpage);
 extern void mem_cgroup_end_migration(struct page *page);
 extern int mem_cgroup_getref(struct page *page);

-/*
```

```
- * 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);
+extern void page_cgroup_init(void);

-#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+#else /* CONFIG_CGROUP_PAGE */
 static inline void page_reset_bad_cgroup(struct page *page)
 {
 }
@@ -98,33 +118,70 @@ 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_getref(struct page *page)
 {
- return 1;
 }
+#endif /* CONFIG_CGROUP_PAGE */

-static inline int
-mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
```

```
+
+#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);
+
+extern struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p);
+
+#define mm_match_cgroup(mm, cgroup) \
+ ((cgroup) == mem_cgroup_from_task((mm)->owner))
+
+/*
+ * 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_getref(struct page *page)
+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)
@@ -163,7 +220,7 @@ static inline long mem_cgroup_calc_recla
 {
  return 0;
 }
-#endif /* CONFIG_CGROUP_MEM_CONT */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */

 #endif /* _LINUX_MEMCONTROL_H */

diff -udpr linux-2.6.26-rc2.cg0/include/linux/mm_types.h linux-2.6.26-rc2/include/linux/mm_types.h
--- linux-2.6.26-rc2.cg0/include/linux/mm_types.h 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/mm_types.h 2008-05-16 19:03:43.000000000 +0900
@@ -91,7 +91,7 @@ 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
diff -udpr linux-2.6.26-rc2.cg0/init/Kconfig linux-2.6.26-rc2/init/Kconfig
--- linux-2.6.26-rc2.cg0/init/Kconfig 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/init/Kconfig 2008-05-16 19:03:43.000000000 +0900
@@ -407,6 +407,10 @@ config CGROUP_MEM_RES_CTLR
    This config option also selects MM_OWNER config option, which
    could in turn add some fork/exit overhead.

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

diff -udpr linux-2.6.26-rc2.cg0/mm/Makefile linux-2.6.26-rc2/mm/Makefile
--- linux-2.6.26-rc2.cg0/mm/Makefile 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/mm/Makefile 2008-05-16 19:03:43.000000000 +0900
@@ -33,5 +33,5 @@ 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

diff -udpr linux-2.6.26-rc2.cg0/mm/memcontrol.c linux-2.6.26-rc2/mm/memcontrol.c
--- linux-2.6.26-rc2.cg0/mm/memcontrol.c 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/mm/memcontrol.c 2008-05-19 11:39:43.000000000 +0900
@@ -35,10 +35,17 @@

 #include <asm/uaccess.h>

-struct cgroup_subsys mem_cgroup_subsys __read_mostly;
 static struct kmem_cache *page_cgroup_cache __read_mostly;
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+struct cgroup_subsys mem_cgroup_subsys __read_mostly;
 #define MEM_CGROUP_RECLAIM_RETRIES 5

+static inline int mem_cgroup_disabled(void)
+{
+ return mem_cgroup_subsys.disabled;
+}
+
 /*
  * Statistics for memory cgroup.
  */
@@ -144,34 +151,6 @@ 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);
@@ -182,11 +161,6 @@ 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()
  */
@@ -254,37 +228,6 @@ struct mem_cgroup *mem_cgroup_from_task(
     struct 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 mem_cgroup_per_zone *mz,
    struct page_cgroup *pc)
 {
@@ -518,252 +461,6 @@ 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 *memcg)
-{
- 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 = kmem_cache_alloc(page_cgroup_cache, 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 (!memcg) {
-  if (!mm)
-   mm = &init_mm;
-
-  rcu_read_lock();
-  mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
-  /*
-   * For every charge from the cgroup, increment reference count
-   */
-  css_get(&mem->css);
-  rcu_read_unlock();
- } else {
-  mem = memcg;
-  css_get(&memcg->css);
- }
-
- 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;
-  }
- }
-
- pc->ref_cnt = 1;
- pc->mem_cgroup = mem;
- pc->page = page;
- /*
-  * If a page is accounted as a page cache, insert to inactive list.
-  * If anon, insert to active list.
-  */
- if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
-  pc->flags = PAGE_CGROUP_FLAG_CACHE;
- else
-  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
-
- 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);
-  kmem_cache_free(page_cgroup_cache, 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(mz, pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- unlock_page_cgroup(page);
```

```
-done:
- return 0;
-out:
- css_put(&mem->css);
- kmem_cache_free(page_cgroup_cache, 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, NULL);
-}
-
-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, NULL);
-}
-
-int mem_cgroup_getref(struct page *page)
-{
- struct page_cgroup *pc;
-
- if (mem_cgroup_subsys.disabled)
-  return 0;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- VM_BUG_ON(!pc);
- pc->ref_cnt++;
- unlock_page_cgroup(page);
- return 0;
-}
-
-/*
- * 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(mz, 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);
-
-  kmem_cache_free(page_cgroup_cache, pc);
-  return;
- }
-
-unlock:
- unlock_page_cgroup(page);
-}
-
-/*
- * Before starting migration, account against new page.
- */
-int mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
-{
- struct page_cgroup *pc;
- struct mem_cgroup *mem = NULL;
- enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
- int ret = 0;
-
- if (mem_cgroup_subsys.disabled)
```

```
-  return 0;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (pc) {
-  mem = pc->mem_cgroup;
-  css_get(&mem->css);
-  if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
-   ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
- }
- unlock_page_cgroup(page);
- if (mem) {
-  ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
-   ctype, mem);
-  css_put(&mem->css);
- }
- return ret;
-}
-
-/* remove redundant charge */
-void mem_cgroup_end_migration(struct page *newpage)
-{
- mem_cgroup_uncharge_page(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.
@@ -817,7 +514,7 @@ 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);
@@ -1033,7 +730,7 @@ mem_cgroup_create(struct cgroup_subsys *

  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
-  page_cgroup_cache = KMEM_CACHE(page_cgroup, SLAB_PANIC);
+  page_cgroup_init();
  } else {
   mem = mem_cgroup_alloc();
   if (!mem)
@@ -1077,7 +774,7 @@ 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));
@@ -1091,7 +788,7 @@ 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);
@@ -1125,3 +822,310 @@ 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 */
+
+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 mem_cgroup *memcg)
+{
+ 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 = kmem_cache_alloc(page_cgroup_cache, 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 (!memcg) {
+  if (!mm)
+   mm = &init_mm;
+
+  rcu_read_lock();
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
+  /*
+   * For every charge from the cgroup, increment reference count
+   */
+  css_get(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  rcu_read_unlock();
+ } else {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = memcg;
+  css_get(&memcg->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ }
+
+#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;
+  }
+ }
+#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;
+ /*
+  * If a page is accounted as a page cache, insert to inactive list.
+  * If anon, insert to active list.
+  */
+ if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
+  pc->flags = PAGE_CGROUP_FLAG_CACHE;
+ else
+  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
+
+ 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 */
+  kmem_cache_free(page_cgroup_cache, 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(mz, 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 */
+ kmem_cache_free(page_cgroup_cache, 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, NULL);
+}
+
+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, NULL);
+}
+
+int mem_cgroup_getref(struct page *page)
+{
+ struct page_cgroup *pc;
+
+ if (mem_cgroup_disabled())
+  return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ VM_BUG_ON(!pc);
+ pc->ref_cnt++;
+ unlock_page_cgroup(page);
+ return 0;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_cgroup_uncharge_page(struct page *page)
+{
+ struct page_cgroup *pc;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct mem_cgroup *mem;
```

```
+ struct mem_cgroup_per_zone *mz;
+ unsigned long flags;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ 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(mz, 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 */
+
+  kmem_cache_free(page_cgroup_cache, pc);
+  return;
+ }
+
+unlock:
+ unlock_page_cgroup(page);
+}
+
+/*
+ * Before starting migration, account against new page.
+ */
+int mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
+{
```

```
+ struct page_cgroup *pc;
+ struct mem_cgroup *mem = NULL;
+ enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
+ int ret = 0;
+
+ if (mem_cgroup_disabled())
+  return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc) {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = pc->mem_cgroup;
+  css_get(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
+   ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
+ }
+ unlock_page_cgroup(page);
+ if (mem) {
+  ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
+   ctype, mem);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ }
+ return ret;
+}
+
+/* remove redundant charge */
+void mem_cgroup_end_migration(struct page *newpage)
+{
+ mem_cgroup_uncharge_page(newpage);
+}
+
+void page_cgroup_init()
+{
+ if (!page_cgroup_cache)
+  page_cgroup_cache = KMEM_CACHE(page_cgroup, SLAB_PANIC);
+}
```

_____

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

---

Subject: [PATCH 2/4] BIO tracking take2

Hi,

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

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

```
diff -dupr linux-2.6.26-rc2.cg1/mm/memcontrol.c linux-2.6.26-rc2/mm/memcontrol.c
--- linux-2.6.26-rc2.cg1/mm/memcontrol.c 2008-05-19 12:33:32.000000000 +0900
+++ linux-2.6.26-rc2/mm/memcontrol.c 2008-05-19 12:34:16.000000000 +0900
@@ -228,6 +228,47 @@ struct mem_cgroup *mem_cgroup_from_task(
     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);
+}
+
+static inline struct mem_cgroup *get_mem_page_cgroup(struct page_cgroup *pc)
+{
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ css_get(&mem->css);
+ return mem;
+}
```

```
+
+/* This sould be called in an RCU-protected section. */
+static inline struct mem_cgroup *mm_get_mem_cgroup(struct mm_struct *mm)
+{
+ struct mem_cgroup *mem;
+
+ mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
+ get_mem_cgroup(mem);
+ return mem;
+}
+
 static void __mem_cgroup_remove_list(struct mem_cgroup_per_zone *mz,
    struct page_cgroup *pc)
 {
@@ -278,6 +319,26 @@ 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(mz, 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(mz, pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+}
+
 int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
 {
  int ret;
@@ -317,6 +378,36 @@ 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;
+  }
+ }
+ return 0;
+}
+
 /*
  * Calculate mapped_ratio under memory controller. This will be used in
  * vmscan.c for deteremining we have to reclaim mapped pages.
@@ -517,7 +608,7 @@ 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.
@@ -538,7 +629,7 @@ static int mem_cgroup_force_empty(struct
  }
  ret = 0;
 out:
- css_put(&mem->css);
+ put_mem_cgroup(mem);
  return ret;
 }
```

```
@@ -825,10 +916,37 @@ struct cgroup_subsys mem_cgroup_subsys =

 #else /* CONFIG_CGROUP_MEM_RES_CTLR */

+struct mem_cgroup;
+
 static inline int mem_cgroup_disabled(void)
 {
  return 1;
 }
+
+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 *get_mem_page_cgroup(struct page_cgroup *pc)
+{
+ return NULL;
+}
+
+static inline struct mem_cgroup *mm_get_mem_cgroup(struct mm_struct *mm)
+{
+ return NULL;
+}
+
+static inline int mem_cgroup_try_to_allocate(struct mem_cgroup *mem,
+      gfp_t gfp_mask)
+{
+ return 0;
+}
+
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR */

 static inline int page_cgroup_locked(struct page *page)
@@ -863,12 +981,7 @@ static int mem_cgroup_charge_common(stru
     struct mem_cgroup *memcg)
 {
  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;
@@ -912,50 +1025,18 @@ retry:
    mm = &init_mm;

   rcu_read_lock();
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
-  mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
-  /*
-   * 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);
   rcu_read_unlock();
  } else {
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
   mem = memcg;
-  css_get(&memcg->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  get_mem_cgroup(mem);
  }

-#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;
-  }
- }
-#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);
  pc->page = page;
  /*
   * If a page is accounted as a page cache, insert to inactive list.
@@ -974,29 +1055,19 @@ 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);
   kmem_cache_free(page_cgroup_cache, 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(mz, pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ mem_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);
  kmem_cache_free(page_cgroup_cache, pc);
 err:
  return -ENOMEM;
@@ -1039,11 +1110,6 @@ int mem_cgroup_getref(struct page *page)
 void mem_cgroup_uncharge_page(struct page *page)
 {
  struct page_cgroup *pc;
```

```
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
- struct mem_cgroup *mem;
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */

  if (mem_cgroup_disabled())
   return;
@@ -1060,21 +1126,12 @@ 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(mz, pc);
-  spin_unlock_irqrestore(&mz->lru_lock, flags);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  mem_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);

   kmem_cache_free(page_cgroup_cache, pc);
   return;
@@ -1100,10 +1157,7 @@ int mem_cgroup_prepare_migration(struct
  lock_page_cgroup(page);
  pc = page_get_page_cgroup(page);
  if (pc) {
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
-  mem = pc->mem_cgroup;
-  css_get(&mem->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  mem = get_mem_page_cgroup(pc);
   if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
    ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
  }
@@ -1111,9 +1165,7 @@ int mem_cgroup_prepare_migration(struct
  if (mem) {
   ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
    ctype, mem);
```

```
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
-  css_put(&mem->css);
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  put_mem_cgroup(mem);
  }
  return ret;
 }
```

_____

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

---

Subject: [PATCH 3/4] BIO tracking take2
Posted by Hirokazu Takahashi on Tue, 20 May 2008 12:03:24 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>

```
diff -dupr linux-2.6.26-rc2.cg2/block/blk-ioc.c linux-2.6.26-rc2/block/blk-ioc.c
--- linux-2.6.26-rc2.cg2/block/blk-ioc.c 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/block/blk-ioc.c 2008-05-19 18:40:10.000000000 +0900
@@ -84,24 +84,28 @@ void exit_io_context(void)
  }
 }

+void init_io_context(struct io_context *ioc)
+{
+ atomic_set(&ioc->refcount, 1);
+ atomic_set(&ioc->nr_tasks, 1);
+ spin_lock_init(&ioc->lock);
+ ioc->ioprio_changed = 0;
+ ioc->ioprio = 0;
+ ioc->last_waited = jiffies; /* doesn't matter... */
+ ioc->nr_batch_requests = 0; /* because this is 0 */
+ ioc->aic = NULL;
+ INIT_RADIX_TREE(&ioc->radix_root, GFP_ATOMIC | __GFP_HIGH);
+ INIT_HLIST_HEAD(&ioc->cic_list);
+ ioc->ioc_data = NULL;
+}
+
 struct io_context *alloc_io_context(gfp_t gfp_flags, int node)
 {
```

```
  struct io_context *ret;

  ret = kmem_cache_alloc_node(iocontext_cachep, gfp_flags, node);
- if (ret) {
-  atomic_set(&ret->refcount, 1);
-  atomic_set(&ret->nr_tasks, 1);
-  spin_lock_init(&ret->lock);
-  ret->ioprio_changed = 0;
-  ret->ioprio = 0;
-  ret->last_waited = jiffies; /* doesn't matter... */
-  ret->nr_batch_requests = 0; /* because this is 0 */
-  ret->aic = NULL;
-  INIT_RADIX_TREE(&ret->radix_root, GFP_ATOMIC | __GFP_HIGH);
-  INIT_HLIST_HEAD(&ret->cic_list);
-  ret->ioc_data = NULL;
- }
+ if (ret)
+  init_io_context(ret);

  return ret;
 }
diff -dupr linux-2.6.26-rc2.cg2/include/linux/biocontrol.h linux-2.6.26-rc2/include/linux/biocontrol.h
--- linux-2.6.26-rc2.cg2/include/linux/biocontrol.h 2008-05-19 13:51:21.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/biocontrol.h 2008-05-19 18:40:10.000000000 +0900
@@ -0,0 +1,160 @@
+#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 block_device;
+
+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;
+}
+
+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);
+}
+
+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);
+}
+
+static inline struct bio_cgroup *get_bio_page_cgroup(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ css_get(&biog->css);
+ return biog;
+}
+
+/* This sould be called in an RCU-protected section. */
+static inline struct bio_cgroup *mm_get_bio_cgroup(struct mm_struct *mm)
+{
+ struct bio_cgroup *biog;
+ biog = bio_cgroup_from_task(rcu_dereference(mm->owner));
+ get_bio_cgroup(biog);
+ return biog;
+}
+
+//extern int get_bio_cgroup_id(struct page *page);
+extern struct io_context *get_bio_cgroup_iocontext(struct bio *bio);
+
+#else /* CONFIG_CGROUP_BIO */
+
+struct bio_cgroup;
+
+static inline int bio_cgroup_disabled(void)
+{
+       return 1;
+}
+
+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 *get_bio_page_cgroup(struct page_cgroup *pc)
+{
+ return NULL;
+}
+
+static inline struct bio_cgroup *mm_get_bio_cgroup(struct mm_struct *mm)
+{
+ return NULL;
+}
+
+static inline int get_bio_cgroup_id(struct page *page)
+{
+ return 0;
+}
+
+static inline struct io_context *get_bio_cgroup_iocontext(struct bio *bio)
+{
+ return NULL;
+}
+
+#endif /* CONFIG_CGROUP_BIO */
+
+#endif /* _LINUX_BIOCONTROL_H */
diff -dupr linux-2.6.26-rc2.cg2/include/linux/blkdev.h linux-2.6.26-rc2/include/linux/blkdev.h
--- linux-2.6.26-rc2.cg2/include/linux/blkdev.h 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/blkdev.h 2008-05-19 18:40:10.000000000 +0900
@@ -38,6 +38,7 @@ int put_io_context(struct io_context *io
 void exit_io_context(void);
 struct io_context *get_io_context(gfp_t gfp_flags, int node);
 struct io_context *alloc_io_context(gfp_t gfp_flags, int node);
```

```
+void init_io_context(struct io_context *ioc);
 void copy_io_context(struct io_context **pdst, struct io_context **psrc);

 struct request;
diff -dupr linux-2.6.26-rc2.cg2/include/linux/cgroup_subsys.h
linux-2.6.26-rc2/include/linux/cgroup_subsys.h
--- linux-2.6.26-rc2.cg2/include/linux/cgroup_subsys.h 2008-05-19 13:51:21.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/cgroup_subsys.h 2008-05-19 18:40:10.000000000 +0900
@@ -43,6 +43,12 @@ SUBSYS(mem_cgroup)

 /* */

+#ifdef CONFIG_CGROUP_BIO
+SUBSYS(bio_cgroup)
+#endif
+
+/* */
+
 #ifdef CONFIG_CGROUP_DEVICE
 SUBSYS(devices)
 #endif
diff -dupr linux-2.6.26-rc2.cg2/include/linux/iocontext.h linux-2.6.26-rc2/include/linux/iocontext.h
--- linux-2.6.26-rc2.cg2/include/linux/iocontext.h 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/iocontext.h 2008-05-19 18:40:10.000000000 +0900
@@ -83,6 +83,8 @@ struct io_context {
  struct radix_tree_root radix_root;
  struct hlist_head cic_list;
  void *ioc_data;
+
+ int id;  /* cgroup ID */
 };

 static inline struct io_context *ioc_task_link(struct io_context *ioc)
diff -dupr linux-2.6.26-rc2.cg2/include/linux/memcontrol.h
linux-2.6.26-rc2/include/linux/memcontrol.h
--- linux-2.6.26-rc2.cg2/include/linux/memcontrol.h 2008-05-19 13:51:21.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/memcontrol.h 2008-05-19 18:40:10.000000000 +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;
```

```
diff -dupr linux-2.6.26-rc2.cg2/init/Kconfig linux-2.6.26-rc2/init/Kconfig
--- linux-2.6.26-rc2.cg2/init/Kconfig 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/init/Kconfig 2008-05-19 18:40:10.000000000 +0900
@@ -407,9 +407,20 @@ config CGROUP_MEM_RES_CTLR
    This config option also selects MM_OWNER config option, which
    could in turn add some fork/exit overhead.

+config CGROUP_BIO
+ bool "Block I/O cgroup subsystem"
+ depends on CGROUPS
+ select MM_OWNER
+ help
+   Provides a Resource Controller which enables to track the onwner
+   of every Block I/O.
+   The information this subsystem provides can be used from any
+   kind of module such as dm-ioband device mapper modules or
+   the cfq-scheduler.
+
 config CGROUP_PAGE
  def_bool y
- depends on CGROUP_MEM_RES_CTLR
+ depends on CGROUP_MEM_RES_CTLR || CGROUP_BIO

 config SYSFS_DEPRECATED
  bool
diff -dupr linux-2.6.26-rc2.cg2/mm/biocontrol.c linux-2.6.26-rc2/mm/biocontrol.c
--- linux-2.6.26-rc2.cg2/mm/biocontrol.c 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/mm/biocontrol.c 2008-05-19 20:51:01.000000000 +0900
@@ -0,0 +1,233 @@
+/* biocontrol.c - Block I/O Controller
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ * 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/blkdev.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 struct idr bio_cgroup_id;
+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;
+ struct io_context *ioc;
+ int error;
+
+ if (!cgrp->parent) {
+  static struct bio_cgroup default_bio_cgroup;
+  static struct io_context default_bio_io_context;
+
+  biog = &default_bio_cgroup;
+  ioc = &default_bio_io_context;
+  init_io_context(ioc);
+
+  idr_init(&bio_cgroup_id);
+  biog->id = 0;
+
+  page_cgroup_init();
+ } else {
+  biog = kzalloc(sizeof(*biog), GFP_KERNEL);
+  ioc = alloc_io_context(GFP_KERNEL, -1);
+  if (!ioc || !biog) {
+   error = -ENOMEM;
+   goto out;
```

```
+  }
+retry:
+  if (unlikely(!idr_pre_get(&bio_cgroup_id, GFP_KERNEL))) {
+   error = -EAGAIN;
+   goto out;
+  }
+  spin_lock_irq(&bio_cgroup_idr_lock);
+  error = idr_get_new_above(&bio_cgroup_id, (void *)biog, 1, &biog->id);
+  spin_unlock_irq(&bio_cgroup_idr_lock);
+  if (error == -EAGAIN)
+   goto retry;
+  else if (error)
+   goto out;
+ }
+
+ ioc->id = biog->id;
+ biog->io_context = ioc;
+
+ 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:
+ if (ioc)
+  put_io_context(ioc);
+ if (biog)
+  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);
+
+ put_io_context(biog->io_context);
+
+       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;
+}
+
+struct io_context *get_bio_cgroup_iocontext(struct bio *bio)
+{
+ struct io_context *ioc;
+ struct page_cgroup *pc;
+ struct bio_cgroup *biog;
+ struct page *page = bio_iovec_idx(bio, 0)->bv_page;
```

```
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc)
+  biog = pc->bio_cgroup;
+ else
+  biog = bio_cgroup_from_task(rcu_dereference(init_mm.owner));
+ ioc = biog->io_context; /* default io_context for this cgroup */
+ atomic_inc(&ioc->refcount);
+ unlock_page_cgroup(page);
+ return ioc;
+}
+EXPORT_SYMBOL(get_bio_cgroup_iocontext);
+
+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);
+
+ 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,
+};
diff -dupr linux-2.6.26-rc2.cg2/mm/Makefile linux-2.6.26-rc2/mm/Makefile
--- linux-2.6.26-rc2.cg2/mm/Makefile 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/mm/Makefile 2008-05-19 18:40:10.000000000 +0900
@@ -34,4 +34,5 @@ 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

diff -dupr linux-2.6.26-rc2.cg2/mm/memcontrol.c linux-2.6.26-rc2/mm/memcontrol.c
--- linux-2.6.26-rc2.cg2/mm/memcontrol.c 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/mm/memcontrol.c 2008-05-19 18:40:10.000000000 +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>
@@ -978,12 +979,13 @@ enum charge_type {
  */
 static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
     gfp_t gfp_mask, enum charge_type ctype,
-    struct mem_cgroup *memcg)
+    struct mem_cgroup *memcg, struct bio_cgroup *biocg)
```

```
 {
  struct page_cgroup *pc;
  struct mem_cgroup *mem;
+ struct bio_cgroup *biog;

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

  /*
@@ -1020,23 +1022,19 @@ retry:
   * thread group leader migrates. It's possible that mm is not
   * set, if so charge the init_mm (happens for pagecache usage).
   */
- if (!memcg) {
-  if (!mm)
-   mm = &init_mm;
-
-  rcu_read_lock();
-  mem = mm_get_mem_cgroup(mm);
-  rcu_read_unlock();
- } else {
-  mem = memcg;
-  get_mem_cgroup(mem);
- }
+ if (!mm)
+  mm = &init_mm;
+ rcu_read_lock();
+ mem = memcg ? memcg : mm_get_mem_cgroup(mm);
+ biog = biocg ? biocg : mm_get_bio_cgroup(mm);
+ rcu_read_unlock();

  if (mem_cgroup_try_to_allocate(mem, gfp_mask) < 0)
   goto out;

  pc->ref_cnt = 1;
  set_mem_cgroup(pc, mem);
+ set_bio_cgroup(pc, biog);
  pc->page = page;
  /*
   * If a page is accounted as a page cache, insert to inactive list.
@@ -1056,18 +1054,21 @@ retry:
    * page->cgroup, increment refcnt.... just retry is OK.
    */
   clear_mem_cgroup(pc);
+  clear_bio_cgroup(pc);
   kmem_cache_free(page_cgroup_cache, pc);
   goto retry;
```

```
  }
  page_assign_page_cgroup(page, pc);

  mem_cgroup_add_page(pc);
+ bio_cgroup_add_page(pc);

  unlock_page_cgroup(page);
 done:
  return 0;
 out:
  put_mem_cgroup(mem);
+ put_bio_cgroup(biog);
  kmem_cache_free(page_cgroup_cache, pc);
 err:
  return -ENOMEM;
@@ -1076,7 +1077,7 @@ err:
 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, NULL);
+    MEM_CGROUP_CHARGE_TYPE_MAPPED, NULL, NULL);
 }

 int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
@@ -1085,7 +1086,7 @@ int mem_cgroup_cache_charge(struct page
   if (!mm)
    mm = &init_mm;
   return mem_cgroup_charge_common(page, mm, gfp_mask,
-    MEM_CGROUP_CHARGE_TYPE_CACHE, NULL);
+    MEM_CGROUP_CHARGE_TYPE_CACHE, NULL, NULL);
 }

 int mem_cgroup_getref(struct page *page)
@@ -1111,7 +1112,7 @@ void mem_cgroup_uncharge_page(struct pag
 {
  struct page_cgroup *pc;

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

  /*
@@ -1127,11 +1128,13 @@ void mem_cgroup_uncharge_page(struct pag

  if (--(pc->ref_cnt) == 0) {
   mem_cgroup_remove_page(pc);
+  bio_cgroup_remove_page(pc);
```

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

   clear_mem_cgroup(pc);
+  clear_bio_cgroup(pc);

   kmem_cache_free(page_cgroup_cache, pc);
   return;
@@ -1148,24 +1151,29 @@ int mem_cgroup_prepare_migration(struct
 {
  struct page_cgroup *pc;
  struct mem_cgroup *mem = NULL;
+ struct bio_cgroup *biog = NULL;
  enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
  int ret = 0;

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

  lock_page_cgroup(page);
  pc = page_get_page_cgroup(page);
  if (pc) {
   mem = get_mem_page_cgroup(pc);
+  biog = get_bio_page_cgroup(pc);
   if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
    ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
  }
  unlock_page_cgroup(page);
- if (mem) {
+ if (pc) {
   ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
-   ctype, mem);
-  put_mem_cgroup(mem);
+   ctype, mem, biog);
+  if (mem)
+   put_mem_cgroup(mem);
+  if (biog)
+   put_bio_cgroup(biog);
  }
  return ret;
 }
```

Subject: [PATCH 4/4] BIO tracking take2
Posted by Hirokazu Takahashi on Tue, 20 May 2008 12:04:05 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>

```
diff -dupr linux-2.6.26-rc2.cg2/drivers/md/dm-ioband-type.c
linux-2.6.26-rc2/drivers/md/dm-ioband-type.c
--- linux-2.6.26-rc2.cg2/drivers/md/dm-ioband-type.c 2008-05-19 13:51:23.000000000 +0900
+++ linux-2.6.26-rc2/drivers/md/dm-ioband-type.c 2008-05-19 18:40:10.000000000 +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"
@@ -53,13 +54,13 @@ 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 */
+ struct io_context *ioc = get_bio_cgroup_iocontext(bio);
+ int id = 0;
+ if (ioc) {
+  id = ioc->id;
+  put_io_context(ioc);
+ }
+ return id;
 }

 struct group_type dm_ioband_group_type[] = {
```

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