Subject: [PATCH 3/3] i/o accounting and control
Posted by Andrea Righi on Fri, 04 Jul 2008 13:58:48 GMT
View Forum Message <> Reply to Message

Apply the io-throttle controller to the opportune kernel functions. Both
accounting and throttling functionalities are performed by
cgroup_io_throttle().

Signed-off-by: Andrea Righi <righi.andrea@gmail.com>
---
 block/blk-core.c | 2 ++
 fs/buffer.c | 10 ++++++++++
 fs/direct-io.c | 3 +++
 mm/page-writeback.c | 20 ++++++++++++++++++++
 mm/readahead.c | 5 +++++
 5 files changed, 40 insertions(+), 0 deletions(-)

diff --git a/block/blk-core.c b/block/blk-core.c
index 1905aab..2ac5463 100644
--- a/block/blk-core.c
+++ b/block/blk-core.c
@@ -26,6 +26,7 @@
 #include <linux/swap.h>
 #include <linux/writeback.h>
 #include <linux/task_io_accounting_ops.h>
+#include <linux/blk-io-throttle.h>
 #include <linux/interrupt.h>
 #include <linux/cpu.h>
 #include <linux/blktrace_api.h>
@@ -1482,6 +1483,7 @@ void submit_bio(int rw, struct bio *bio)
    count_vm_events(PGPGOUT, count);
   } else {
    task_io_account_read(bio->bi_size);
+   cgroup_io_throttle(bio->bi_bdev, bio->bi_size, 1);
    count_vm_events(PGPGIN, count);
   }

diff --git a/fs/buffer.c b/fs/buffer.c
index 0f51c0f..6f41c3a 100644
--- a/fs/buffer.c
+++ b/fs/buffer.c
@@ -35,6 +35,7 @@
 #include <linux/suspend.h>
 #include <linux/buffer_head.h>
 #include <linux/task_io_accounting_ops.h>
+#include <linux/blk-io-throttle.h>
 #include <linux/bio.h>
 #include <linux/notifier.h>

```
 #include <linux/cpu.h>
@@ -700,6 +701,9 @@ EXPORT_SYMBOL(mark_buffer_dirty_inode);
 static int __set_page_dirty(struct page *page,
   struct address_space *mapping, int warn)
 {
+ struct block_device *bdev = NULL;
+ size_t cgroup_io_acct = 0;
+
  if (unlikely(!mapping))
   return !TestSetPageDirty(page);

@@ -711,16 +715,22 @@ static int __set_page_dirty(struct page *page,
   WARN_ON_ONCE(warn && !PageUptodate(page));

   if (mapping_cap_account_dirty(mapping)) {
+   bdev = (mapping->host &&
+    mapping->host->i_sb->s_bdev) ?
+    mapping->host->i_sb->s_bdev : NULL;
+
    __inc_zone_page_state(page, NR_FILE_DIRTY);
    __inc_bdi_stat(mapping->backing_dev_info,
     BDI_RECLAIMABLE);
    task_io_account_write(PAGE_CACHE_SIZE);
+   cgroup_io_acct = PAGE_CACHE_SIZE;
   }
   radix_tree_tag_set(&mapping->page_tree,
    page_index(page), PAGECACHE_TAG_DIRTY);
  }
  write_unlock_irq(&mapping->tree_lock);
  __mark_inode_dirty(mapping->host, I_DIRTY_PAGES);
+ cgroup_io_throttle(bdev, cgroup_io_acct, 0);

  return 1;
 }
diff --git a/fs/direct-io.c b/fs/direct-io.c
index 9e81add..42c8e54 100644
--- a/fs/direct-io.c
+++ b/fs/direct-io.c
@@ -35,6 +35,7 @@
 #include <linux/buffer_head.h>
 #include <linux/rwsem.h>
 #include <linux/uio.h>
+#include <linux/blk-io-throttle.h>
 #include <asm/atomic.h>

 /*
@@ -666,6 +667,8 @@ submit_page_section(struct dio *dio, struct page *page,
   /*
```

```
    * Read accounting is performed in submit_bio()
    */
+  struct block_device *bdev = dio->bio ? dio->bio->bi_bdev : NULL;
+  cgroup_io_throttle(bdev, len, 1);
   task_io_account_write(len);
  }

diff --git a/mm/page-writeback.c b/mm/page-writeback.c
index 789b6ad..8e5e99d 100644
--- a/mm/page-writeback.c
+++ b/mm/page-writeback.c
@@ -23,6 +23,7 @@
 #include <linux/init.h>
 #include <linux/backing-dev.h>
 #include <linux/task_io_accounting_ops.h>
+#include <linux/blk-io-throttle.h>
 #include <linux/blkdev.h>
 #include <linux/mpage.h>
 #include <linux/rmap.h>
@@ -430,6 +431,9 @@ static void balance_dirty_pages(struct address_space *mapping)
  unsigned long write_chunk = sync_writeback_pages();

  struct backing_dev_info *bdi = mapping->backing_dev_info;
+ struct block_device *bdev = (mapping->host &&
+     mapping->host->i_sb->s_bdev) ?
+     mapping->host->i_sb->s_bdev : NULL;

  for (;;) {
   struct writeback_control wbc = {
@@ -512,6 +516,14 @@ static void balance_dirty_pages(struct address_space *mapping)
   return;  /* pdflush is already working this queue */

  /*
+  * Apply the cgroup i/o throttling limitations. The accounting of write
+  * activity in page cache is performed in __set_page_dirty(), but since
+  * we cannot sleep there, 0 bytes are accounted here and the function
+  * is invoked only for throttling purpose.
+  */
+ cgroup_io_throttle(bdev, 0, 1);
+
+ /*
   * In laptop mode, we wait until hitting the higher threshold before
   * starting background writeout, and then write out all the way down
   * to the lower threshold.  So slow writers cause minimal disk activity.
@@ -1077,6 +1089,8 @@ int __set_page_dirty_nobuffers(struct page *page)
  if (!TestSetPageDirty(page)) {
   struct address_space *mapping = page_mapping(page);
   struct address_space *mapping2;
```

```
+  struct block_device *bdev = NULL;
+  size_t cgroup_io_acct = 0;

   if (!mapping)
    return 1;
@@ -1087,10 +1101,15 @@ int __set_page_dirty_nobuffers(struct page *page)
    BUG_ON(mapping2 != mapping);
    WARN_ON_ONCE(!PagePrivate(page) && !PageUptodate(page));
    if (mapping_cap_account_dirty(mapping)) {
+    bdev = (mapping->host &&
+     mapping->host->i_sb->s_bdev) ?
+     mapping->host->i_sb->s_bdev : NULL;
+
     __inc_zone_page_state(page, NR_FILE_DIRTY);
     __inc_bdi_stat(mapping->backing_dev_info,
       BDI_RECLAIMABLE);
     task_io_account_write(PAGE_CACHE_SIZE);
+    cgroup_io_acct = PAGE_CACHE_SIZE;
    }
    radix_tree_tag_set(&mapping->page_tree,
     page_index(page), PAGECACHE_TAG_DIRTY);
@@ -1100,6 +1119,7 @@ int __set_page_dirty_nobuffers(struct page *page)
    /* !PageAnon && !swapper_space */
    __mark_inode_dirty(mapping->host, I_DIRTY_PAGES);
   }
+  cgroup_io_throttle(bdev, cgroup_io_acct, 1);
   return 1;
  }
  return 0;
diff --git a/mm/readahead.c b/mm/readahead.c
index d8723a5..ec83e63 100644
--- a/mm/readahead.c
+++ b/mm/readahead.c
@@ -14,6 +14,7 @@
 #include <linux/blkdev.h>
 #include <linux/backing-dev.h>
 #include <linux/task_io_accounting_ops.h>
+#include <linux/blk-io-throttle.h>
 #include <linux/pagevec.h>
 #include <linux/pagemap.h>

@@ -58,6 +59,9 @@ int read_cache_pages(struct address_space *mapping, struct list_head
*pages,
    int (*filler)(void *, struct page *), void *data)
 {
  struct page *page;
+ struct block_device *bdev =
+  (mapping->host && mapping->host->i_sb->s_bdev) ?
```

```
+  mapping->host->i_sb->s_bdev : NULL;
  int ret = 0;

  while (!list_empty(pages)) {
@@ -76,6 +80,7 @@ int read_cache_pages(struct address_space *mapping, struct list_head
*pages,
    break;
   }
   task_io_account_read(PAGE_CACHE_SIZE);
+  cgroup_io_throttle(bdev, PAGE_CACHE_SIZE, 1);
  }
  return ret;
 }
--
1.5.4.3
```

_____

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