Subject: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Andrea Righi on Tue, 22 Feb 2011 17:12:55 GMT
View Forum Message <> Reply to Message

Add the tracking of buffered (writeback) and anonymous pages.

Dirty pages in the page cache can be processed asynchronously by the per-bdi flusher kernel threads or by any other thread in the system, according to the writeback policy.

For this reason the real writes to the underlying block devices may occur in a different IO context respect to the task that originally generated the dirty pages involved in the IO operation. This makes the tracking and throttling of writeback IO more complicate respect to the synchronous IO from the blkio controller's point of view.

The idea is to save the cgroup owner of each anonymous page and dirty page in page cache. A page is associated to a cgroup the first time it is dirtied in memory (for file cache pages) or when it is set as swap-backed (for anonymous pages). This information is stored using the page_cgroup functionality.

Then, at the block layer, it is possible to retrieve the throttle group looking at the bio_page(bio). If the page was not explicitly associated to any cgroup the IO operation is charged to the current task/cgroup, as it was done by the previous implementation.

Signed-off-by: Andrea Righi <arighi@develer.com>
---
 block/blk-throttle.c | 87 ++-
 include/linux/blkdev.h | 26 +++++++++++++-
 2 files changed, 111 insertions(+), 2 deletions(-)

diff --git a/block/blk-throttle.c b/block/blk-throttle.c
index 9ad3d1e..a50ee04 100644
--- a/block/blk-throttle.c
+++ b/block/blk-throttle.c
@@ -8,6 +8,10 @@
 #include <linux/slab.h>
 #include <linux/blkdev.h>
 #include <linux/bio.h>
+#include <linux/memcontrol.h>
+#include <linux/mm_inline.h>
+#include <linux/pagemap.h>
+#include <linux/page_cgroup.h>
 #include <linux/blktrace_api.h>
 #include <linux/blk-cgroup.h>

```
@@ -221,6 +225,85 @@ done:
  return tg;
 }

+static inline bool is_kernel_io(void)
+{
+ return !!(current->flags & (PF_KTHREAD | PF_KSWAPD | PF_MEMALLOC));
+}
+
+static int throtl_set_page_owner(struct page *page, struct mm_struct *mm)
+{
+ struct blkio_cgroup *blkcg;
+ unsigned short id = 0;
+
+ if (blkio_cgroup_disabled())
+  return 0;
+ if (!mm)
+  goto out;
+ rcu_read_lock();
+ blkcg = task_to_blkio_cgroup(rcu_dereference(mm->owner));
+ if (likely(blkcg))
+  id = css_id(&blkcg->css);
+ rcu_read_unlock();
+out:
+ return page_cgroup_set_owner(page, id);
+}
+
+int blk_throtl_set_anonpage_owner(struct page *page, struct mm_struct *mm)
+{
+ return throtl_set_page_owner(page, mm);
+}
+EXPORT_SYMBOL(blk_throtl_set_anonpage_owner);
+
+int blk_throtl_set_filepage_owner(struct page *page, struct mm_struct *mm)
+{
+ if (is_kernel_io() || !page_is_file_cache(page))
+  return 0;
+ return throtl_set_page_owner(page, mm);
+}
+EXPORT_SYMBOL(blk_throtl_set_filepage_owner);
+
+int blk_throtl_copy_page_owner(struct page *npage, struct page *opage)
+{
+ if (blkio_cgroup_disabled())
+  return 0;
+ return page_cgroup_copy_owner(npage, opage);
+}
+EXPORT_SYMBOL(blk_throtl_copy_page_owner);
```

```
+
+/*
+ * A helper function to get the throttle group from css id.
+ *
+ * NOTE: must be called under rcu_read_lock().
+ */
+static struct throtl_grp *throtl_tg_lookup(struct throtl_data *td, int id)
+{
+ struct cgroup_subsys_state *css;
+
+ if (!id)
+  return NULL;
+ css = css_lookup(&blkio_subsys, id);
+ if (!css)
+  return NULL;
+ return throtl_find_alloc_tg(td, css->cgroup);
+}
+
+static struct throtl_grp *
+throtl_get_tg_from_page(struct throtl_data *td, struct page *page)
+{
+ struct throtl_grp *tg;
+ int id;
+
+ if (unlikely(!page))
+  return NULL;
+ id = page_cgroup_get_owner(page);
+
+ rcu_read_lock();
+ tg = throtl_tg_lookup(td, id);
+ rcu_read_unlock();
+
+ return tg;
+}
+
 static struct throtl_grp * throtl_get_tg(struct throtl_data *td)
 {
  struct cgroup *cgroup;
@@ -1000,7 +1083,9 @@ int blk_throtl_bio(struct request_queue *q, struct bio **biop)
  }

  spin_lock_irq(q->queue_lock);
- tg = throtl_get_tg(td);
+ tg = throtl_get_tg_from_page(td, bio_page(bio));
+ if (!tg)
+  tg = throtl_get_tg(td);

  if (tg->nr_queued[rw]) {
```

```
   /*
diff --git a/include/linux/blkdev.h b/include/linux/blkdev.h
index 4d18ff3..2d03dee 100644
--- a/include/linux/blkdev.h
+++ b/include/linux/blkdev.h
@@ -1136,10 +1136,34 @@ static inline uint64_t rq_io_start_time_ns(struct request *req)
 extern int blk_throtl_init(struct request_queue *q);
 extern void blk_throtl_exit(struct request_queue *q);
 extern int blk_throtl_bio(struct request_queue *q, struct bio **bio);
+extern int blk_throtl_set_anonpage_owner(struct page *page,
+    struct mm_struct *mm);
+extern int blk_throtl_set_filepage_owner(struct page *page,
+    struct mm_struct *mm);
+extern int blk_throtl_copy_page_owner(struct page *npage, struct page *opage);
 extern void throtl_schedule_delayed_work(struct request_queue *q, unsigned long delay);
 extern void throtl_shutdown_timer_wq(struct request_queue *q);
 #else /* CONFIG_BLK_DEV_THROTTLING */
-static inline int blk_throtl_bio(struct request_queue *q, struct bio **bio)
+static inline int
+blk_throtl_bio(struct request_queue *q, struct bio **bio)
+{
+ return 0;
+}
+
+static inline int
+blk_throtl_set_anonpage_owner(struct page *page, struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline int
+blk_throtl_set_filepage_owner(struct page *page, struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline int
+blk_throtl_copy_page_owner(struct page *npage, struct page *opage)
 {
  return 0;
 }
--
1.7.1
```

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