Subject: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Andrea Righi on Tue, 22 Feb 2011 17:12:51 GMT
View Forum Message <> Reply to Message

Currently the blkio.throttle controller only support synchronous IO requests.
This means that we always look at the current task to identify the "owner" of
each IO request.

However dirty pages in the page cache can be wrote to disk 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 perspective.

The same concept is also valid for anonymous pages involed in IO operations
(swap).

This patch allow to track the cgroup that originally dirtied each page in page
cache and each anonymous page and pass these informations to the blk-throttle
controller. These informations can be used to provide a better service level
differentiation of buffered writes swap IO between different cgroups.

Testcase
========
- create a cgroup with 1MiB/s write limit:
  # mount -t cgroup -o blkio none /mnt/cgroup
  # mkdir /mnt/cgroup/foo
  # echo 8:0 $((1024 * 1024)) > /mnt/cgroup/foo/blkio.throttle.write_bps_device

- move a task into the cgroup and run a dd to generate some writeback IO

Results:
  - 2.6.38-rc6 vanilla:
  $ cat /proc/$$/cgroup
  1:blkio:/foo
  $ dd if=/dev/zero of=zero bs=1M count=1024 &
  $ dstat -df
  --dsk/sda--
   read writ
     0   19M
     0   19M
     0   0
     0   0
     0   19M

...

- 2.6.38-rc6 + blk-throttle writeback IO control:
$ cat /proc/$$/cgroup
1:blkio:/foo
$ dd if=/dev/zero of=zero bs=1M count=1024 &
$ dstat -df
--dsk/sda--
 read writ
   0 1024
   0 1024
   0 1024
   0 1024
   0 1024
...

TODO
====
- lots of testing

Any feedback is welcome.
-Andrea

[PATCH 1/5] blk-cgroup: move blk-cgroup.h in include/linux/blk-cgroup.h
[PATCH 2/5] blk-cgroup: introduce task_to_blkio_cgroup()
[PATCH 3/5] page_cgroup: make page tracking available for blkio
[PATCH 4/5] blk-throttle: track buffered and anonymous pages
[PATCH 5/5] blk-throttle: buffered and anonymous page tracking instrumentation

 block/Kconfig | 2 +
 block/blk-cgroup.c | 15 ++-
 block/blk-cgroup.h | 335 --
 block/blk-throttle.c | 89 ++++++++++-
 block/cfq.h | 2 +-
 fs/buffer.c | 1 +
 include/linux/blk-cgroup.h | 341 ++
 include/linux/blkdev.h | 26 +++-
 include/linux/memcontrol.h | 6 +
 include/linux/mmzone.h | 4 +-
 include/linux/page_cgroup.h | 33 ++++-
 init/Kconfig | 4 +
 mm/Makefile | 3 +-
 mm/bounce.c | 1 +
 mm/filemap.c | 1 +
 mm/memcontrol.c | 6 +
 mm/memory.c | 5 +
 mm/page-writeback.c | 1 +
 mm/page_cgroup.c | 129 ++++++++++++++++--

```
 mm/swap_state.c            |    2 +
 20 files changed, 649 insertions(+), 357 deletions(-)
```

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

---

Subject: [PATCH 1/5] blk-cgroup: move blk-cgroup.h in include/linux/blk-cgroup.h
Posted by Andrea Righi on Tue, 22 Feb 2011 17:12:52 GMT
View Forum Message <> Reply to Message

Move blk-cgroup.h in include/linux for generic usage.

Signed-off-by: Andrea Righi <arighi@develer.com>

```
---
 block/blk-cgroup.c         |    2 +-
 block/blk-cgroup.h         |  335 ------------------------------------------
 block/blk-throttle.c       |    2 +-
 block/cfq.h                |    2 +-
 include/linux/blk-cgroup.h |  337 ++++++++++++++++++++++++++++++++++++++++++++
 5 files changed, 340 insertions(+), 338 deletions(-)
 delete mode 100644 block/blk-cgroup.h
 create mode 100644 include/linux/blk-cgroup.h

diff --git a/block/blk-cgroup.c b/block/blk-cgroup.c
index 455768a..bf9d354 100644
--- a/block/blk-cgroup.c
+++ b/block/blk-cgroup.c
@@ -17,7 +17,7 @@
 #include <linux/err.h>
 #include <linux/blkdev.h>
 #include <linux/slab.h>
-#include "blk-cgroup.h"
+#include <linux/blk-cgroup.h>
 #include <linux/genhd.h>

 #define MAX_KEY_LEN 100
diff --git a/block/blk-cgroup.h b/block/blk-cgroup.h
deleted file mode 100644
index ea4861b..0000000
--- a/block/blk-cgroup.h
+++ /dev/null
@@ -1,335 +0,0 @@
-#ifndef _BLK_CGROUP_H
-#define _BLK_CGROUP_H
-/*
- * Common Block IO controller cgroup interface
```

```
- *
- * Based on ideas and code from CFQ, CFS and BFQ:
- * Copyright (C) 2003 Jens Axboe <axboe@kernel.dk>
- *
- * Copyright (C) 2008 Fabio Checconi <fabio@gandalf.sssup.it>
- *        Paolo Valente <paolo.valente@unimore.it>
- *
- * Copyright (C) 2009 Vivek Goyal <vgoyal@redhat.com>
- *                Nauman Rafique <nauman@google.com>
- */
-
-#include <linux/cgroup.h>
-
-enum blkio_policy_id {
- BLKIO_POLICY_PROP = 0,  /* Proportional Bandwidth division */
- BLKIO_POLICY_THROTL,  /* Throttling */
-};
-
-/* Max limits for throttle policy */
-#define THROTL_IOPS_MAX  UINT_MAX
-
-#if defined(CONFIG_BLK_CGROUP) || defined(CONFIG_BLK_CGROUP_MODULE)
-
-#ifndef CONFIG_BLK_CGROUP
-/* When blk-cgroup is a module, its subsys_id isn't a compile-time constant */
-extern struct cgroup_subsys blkio_subsys;
-#define blkio_subsys_id blkio_subsys.subsys_id
-#endif
-
-enum stat_type {
- /* Total time spent (in ns) between request dispatch to the driver and
-  * request completion for IOs doen by this cgroup. This may not be
-  * accurate when NCQ is turned on. */
- BLKIO_STAT_SERVICE_TIME = 0,
- /* Total bytes transferred */
- BLKIO_STAT_SERVICE_BYTES,
- /* Total IOs serviced, post merge */
- BLKIO_STAT_SERVICED,
- /* Total time spent waiting in scheduler queue in ns */
- BLKIO_STAT_WAIT_TIME,
- /* Number of IOs merged */
- BLKIO_STAT_MERGED,
- /* Number of IOs queued up */
- BLKIO_STAT_QUEUED,
- /* All the single valued stats go below this */
- BLKIO_STAT_TIME,
- BLKIO_STAT_SECTORS,
-#ifdef CONFIG_DEBUG_BLK_CGROUP
```

- BLKIO_STAT_AVG_QUEUE_SIZE,
- BLKIO_STAT_IDLE_TIME,
- BLKIO_STAT_EMPTY_TIME,
- BLKIO_STAT_GROUP_WAIT_TIME,
- BLKIO_STAT_DEQUEUE
-#endif
-};
-
-enum stat_sub_type {
- BLKIO_STAT_READ = 0,
- BLKIO_STAT_WRITE,
- BLKIO_STAT_SYNC,
- BLKIO_STAT_ASYNC,
- BLKIO_STAT_TOTAL
-};
-
-/* blkg state flags */
-enum blkg_state_flags {
- BLKG_waiting = 0,
- BLKG_idling,
- BLKG_empty,
-};
-
-/* cgroup files owned by proportional weight policy */
-enum blkcg_file_name_prop {
- BLKIO_PROP_weight = 1,
- BLKIO_PROP_weight_device,
- BLKIO_PROP_io_service_bytes,
- BLKIO_PROP_io_serviced,
- BLKIO_PROP_time,
- BLKIO_PROP_sectors,
- BLKIO_PROP_io_service_time,
- BLKIO_PROP_io_wait_time,
- BLKIO_PROP_io_merged,
- BLKIO_PROP_io_queued,
- BLKIO_PROP_avg_queue_size,
- BLKIO_PROP_group_wait_time,
- BLKIO_PROP_idle_time,
- BLKIO_PROP_empty_time,
- BLKIO_PROP_dequeue,
-};
-
-/* cgroup files owned by throttle policy */
-enum blkcg_file_name_throtl {
- BLKIO_THROTL_read_bps_device,
- BLKIO_THROTL_write_bps_device,
- BLKIO_THROTL_read_iops_device,
- BLKIO_THROTL_write_iops_device,

- BLKIO_THROTL_io_service_bytes,
- BLKIO_THROTL_io_serviced,
-};
-
-struct blkio_cgroup {
- struct cgroup_subsys_state css;
- unsigned int weight;
- spinlock_t lock;
- struct hlist_head blkg_list;
- struct list_head policy_list; /* list of blkio_policy_node */
-};
-
-struct blkio_group_stats {
- /* total disk time and nr sectors dispatched by this group */
- uint64_t time;
- uint64_t sectors;
- uint64_t stat_arr[BLKIO_STAT_QUEUED + 1][BLKIO_STAT_TOTAL];
-#ifdef CONFIG_DEBUG_BLK_CGROUP
- /* Sum of number of IOs queued across all samples */
- uint64_t avg_queue_size_sum;
- /* Count of samples taken for average */
- uint64_t avg_queue_size_samples;
- /* How many times this group has been removed from service tree */
- unsigned long dequeue;
-
- /* Total time spent waiting for it to be assigned a timeslice. */
- uint64_t group_wait_time;
- uint64_t start_group_wait_time;
-
- /* Time spent idling for this blkio_group */
- uint64_t idle_time;
- uint64_t start_idle_time;
- /*
- * Total time when we have requests queued and do not contain the
- * current active queue.
- */
- uint64_t empty_time;
- uint64_t start_empty_time;
- uint16_t flags;
-#endif
-};
-
-struct blkio_group {
- /* An rcu protected unique identifier for the group */
- void *key;
- struct hlist_node blkcg_node;
- unsigned short blkcg_id;
- /* Store cgroup path */

```
- char path[128];
- /* The device MKDEV(major, minor), this group has been created for */
- dev_t dev;
- /* policy which owns this blk group */
- enum blkio_policy_id plid;
-
- /* Need to serialize the stats in the case of reset/update */
- spinlock_t stats_lock;
- struct blkio_group_stats stats;
-};
-
-struct blkio_policy_node {
- struct list_head node;
- dev_t dev;
- /* This node belongs to max bw policy or porportional weight policy */
- enum blkio_policy_id plid;
- /* cgroup file to which this rule belongs to */
- int fileid;
-
- union {
-  unsigned int weight;
-  /*
-   * Rate read/write in terms of byptes per second
-   * Whether this rate represents read or write is determined
-   * by file type "fileid".
-   */
-  u64 bps;
-  unsigned int iops;
- } val;
-};
-
-extern unsigned int blkcg_get_weight(struct blkio_cgroup *blkcg,
-         dev_t dev);
-extern uint64_t blkcg_get_read_bps(struct blkio_cgroup *blkcg,
-         dev_t dev);
-extern uint64_t blkcg_get_write_bps(struct blkio_cgroup *blkcg,
-         dev_t dev);
-extern unsigned int blkcg_get_read_iops(struct blkio_cgroup *blkcg,
-         dev_t dev);
-extern unsigned int blkcg_get_write_iops(struct blkio_cgroup *blkcg,
-         dev_t dev);
-
-typedef void (blkio_unlink_group_fn) (void *key, struct blkio_group *blkg);
-
-typedef void (blkio_update_group_weight_fn) (void *key,
-   struct blkio_group *blkg, unsigned int weight);
-typedef void (blkio_update_group_read_bps_fn) (void * key,
-   struct blkio_group *blkg, u64 read_bps);
```

```
-typedef void (blkio_update_group_write_bps_fn) (void *key,
-   struct blkio_group *blkg, u64 write_bps);
-typedef void (blkio_update_group_read_iops_fn) (void *key,
-   struct blkio_group *blkg, unsigned int read_iops);
-typedef void (blkio_update_group_write_iops_fn) (void *key,
-   struct blkio_group *blkg, unsigned int write_iops);
-
-struct blkio_policy_ops {
- blkio_unlink_group_fn *blkio_unlink_group_fn;
- blkio_update_group_weight_fn *blkio_update_group_weight_fn;
- blkio_update_group_read_bps_fn *blkio_update_group_read_bps_fn;
- blkio_update_group_write_bps_fn *blkio_update_group_write_bps_fn;
- blkio_update_group_read_iops_fn *blkio_update_group_read_iops_fn;
- blkio_update_group_write_iops_fn *blkio_update_group_write_iops_fn;
-};
-
-struct blkio_policy_type {
- struct list_head list;
- struct blkio_policy_ops ops;
- enum blkio_policy_id plid;
-};
-
-/* Blkio controller policy registration */
-extern void blkio_policy_register(struct blkio_policy_type *);
-extern void blkio_policy_unregister(struct blkio_policy_type *);
-
-static inline char *blkg_path(struct blkio_group *blkg)
-{
- return blkg->path;
-}
-
-#else
-
-struct blkio_group {
-};
-
-struct blkio_policy_type {
-};
-
-static inline void blkio_policy_register(struct blkio_policy_type *blkiop) { }
-static inline void blkio_policy_unregister(struct blkio_policy_type *blkiop) { }
-
-static inline char *blkg_path(struct blkio_group *blkg) { return NULL; }
-
-#endif
-
-#define BLKIO_WEIGHT_MIN 100
-#define BLKIO_WEIGHT_MAX 1000
```

```
-#define BLKIO_WEIGHT_DEFAULT 500
-
-#ifdef CONFIG_DEBUG_BLK_CGROUP
-void blkiocg_update_avg_queue_size_stats(struct blkio_group *blkg);
-void blkiocg_update_dequeue_stats(struct blkio_group *blkg,
-    unsigned long dequeue);
-void blkiocg_update_set_idle_time_stats(struct blkio_group *blkg);
-void blkiocg_update_idle_time_stats(struct blkio_group *blkg);
-void blkiocg_set_start_empty_time(struct blkio_group *blkg);
-
-#define BLKG_FLAG_FNS(name)      \
-static inline void blkio_mark_blkg_##name(    \
-  struct blkio_group_stats *stats)   \
-{        \
- stats->flags |= (1 << BLKG_##name);    \
-}        \
-static inline void blkio_clear_blkg_##name(    \
-  struct blkio_group_stats *stats)   \
-{        \
- stats->flags &= ~(1 << BLKG_##name);    \
-}        \
-static inline int blkio_blkg_##name(struct blkio_group_stats *stats) \
-{        \
- return (stats->flags & (1 << BLKG_##name)) != 0;  \
-}        \
-
-BLKG_FLAG_FNS(waiting)
-BLKG_FLAG_FNS(idling)
-BLKG_FLAG_FNS(empty)
-#undef BLKG_FLAG_FNS
-#else
-static inline void blkiocg_update_avg_queue_size_stats(
-      struct blkio_group *blkg) {}
-static inline void blkiocg_update_dequeue_stats(struct blkio_group *blkg,
-      unsigned long dequeue) {}
-static inline void blkiocg_update_set_idle_time_stats(struct blkio_group *blkg)
-{}
-static inline void blkiocg_update_idle_time_stats(struct blkio_group *blkg) {}
-static inline void blkiocg_set_start_empty_time(struct blkio_group *blkg) {}
-#endif
-
-#if defined(CONFIG_BLK_CGROUP) || defined(CONFIG_BLK_CGROUP_MODULE)
-extern struct blkio_cgroup blkio_root_cgroup;
-extern struct blkio_cgroup *cgroup_to_blkio_cgroup(struct cgroup *cgroup);
-extern void blkiocg_add_blkio_group(struct blkio_cgroup *blkcg,
- struct blkio_group *blkg, void *key, dev_t dev,
- enum blkio_policy_id plid);
-extern int blkiocg_del_blkio_group(struct blkio_group *blkg);
```

```
-extern struct blkio_group *blkiocg_lookup_group(struct blkio_cgroup *blkcg,
-      void *key);
-void blkiocg_update_timeslice_used(struct blkio_group *blkg,
-     unsigned long time);
-void blkiocg_update_dispatch_stats(struct blkio_group *blkg, uint64_t bytes,
-      bool direction, bool sync);
-void blkiocg_update_completion_stats(struct blkio_group *blkg,
- uint64_t start_time, uint64_t io_start_time, bool direction, bool sync);
-void blkiocg_update_io_merged_stats(struct blkio_group *blkg, bool direction,
-     bool sync);
-void blkiocg_update_io_add_stats(struct blkio_group *blkg,
-  struct blkio_group *curr_blkg, bool direction, bool sync);
-void blkiocg_update_io_remove_stats(struct blkio_group *blkg,
-     bool direction, bool sync);
-#else
-struct cgroup;
-static inline struct blkio_cgroup *
-cgroup_to_blkio_cgroup(struct cgroup *cgroup) { return NULL; }
-
-static inline void blkiocg_add_blkio_group(struct blkio_cgroup *blkcg,
-  struct blkio_group *blkg, void *key, dev_t dev,
-  enum blkio_policy_id plid) {}
-
-static inline int
-blkiocg_del_blkio_group(struct blkio_group *blkg) { return 0; }
-
-static inline struct blkio_group *
-blkiocg_lookup_group(struct blkio_cgroup *blkcg, void *key) { return NULL; }
-static inline void blkiocg_update_timeslice_used(struct blkio_group *blkg,
-      unsigned long time) {}
-static inline void blkiocg_update_dispatch_stats(struct blkio_group *blkg,
-    uint64_t bytes, bool direction, bool sync) {}
-static inline void blkiocg_update_completion_stats(struct blkio_group *blkg,
-  uint64_t start_time, uint64_t io_start_time, bool direction,
-  bool sync) {}
-static inline void blkiocg_update_io_merged_stats(struct blkio_group *blkg,
-      bool direction, bool sync) {}
-static inline void blkiocg_update_io_add_stats(struct blkio_group *blkg,
-  struct blkio_group *curr_blkg, bool direction, bool sync) {}
-static inline void blkiocg_update_io_remove_stats(struct blkio_group *blkg,
-      bool direction, bool sync) {}
-#endif
-#endif /* _BLK_CGROUP_H */
diff --git a/block/blk-throttle.c b/block/blk-throttle.c
index a89043a..9ad3d1e 100644
--- a/block/blk-throttle.c
+++ b/block/blk-throttle.c
@@ -9,7 +9,7 @@
```

```
 #include <linux/blkdev.h>
 #include <linux/bio.h>
 #include <linux/blktrace_api.h>
-#include "blk-cgroup.h"
+#include <linux/blk-cgroup.h>

 /* Max dispatch from a group in 1 round */
 static int throtl_grp_quantum = 8;
diff --git a/block/cfq.h b/block/cfq.h
index 54a6d90..1213f9b 100644
--- a/block/cfq.h
+++ b/block/cfq.h
@@ -1,6 +1,6 @@
 #ifndef _CFQ_H
 #define _CFQ_H
-#include "blk-cgroup.h"
+#include <linux/blk-cgroup.h>

 #ifdef CONFIG_CFQ_GROUP_IOSCHED
 static inline void cfq_blkiocg_update_io_add_stats(struct blkio_group *blkg,
diff --git a/include/linux/blk-cgroup.h b/include/linux/blk-cgroup.h
new file mode 100644
index 0000000..5e48204
--- /dev/null
+++ b/include/linux/blk-cgroup.h
@@ -0,0 +1,337 @@
+#ifndef _BLK_CGROUP_H
+#define _BLK_CGROUP_H
+/*
+ * Common Block IO controller cgroup interface
+ *
+ * Based on ideas and code from CFQ, CFS and BFQ:
+ * Copyright (C) 2003 Jens Axboe <axboe@kernel.dk>
+ *
+ * Copyright (C) 2008 Fabio Checconi <fabio@gandalf.sssup.it>
+ *                    Paolo Valente <paolo.valente@unimore.it>
+ *
+ * Copyright (C) 2009 Vivek Goyal <vgoyal@redhat.com>
+ *                    Nauman Rafique <nauman@google.com>
+ */
+
+#include <linux/cgroup.h>
+
+enum blkio_policy_id {
+ BLKIO_POLICY_PROP = 0,  /* Proportional Bandwidth division */
+ BLKIO_POLICY_THROTL,  /* Throttling */
+};
+
```

```
+/* Max limits for throttle policy */
+#define THROTL_IOPS_MAX  UINT_MAX
+
+#if defined(CONFIG_BLK_CGROUP) || defined(CONFIG_BLK_CGROUP_MODULE)
+
+#ifndef CONFIG_BLK_CGROUP
+/* When blk-cgroup is a module, its subsys_id isn't a compile-time constant */
+extern struct cgroup_subsys blkio_subsys;
+#define blkio_subsys_id blkio_subsys.subsys_id
+#endif
+
+enum stat_type {
+ /* Total time spent (in ns) between request dispatch to the driver and
+  * request completion for IOs doen by this cgroup. This may not be
+  * accurate when NCQ is turned on. */
+ BLKIO_STAT_SERVICE_TIME = 0,
+ /* Total bytes transferred */
+ BLKIO_STAT_SERVICE_BYTES,
+ /* Total IOs serviced, post merge */
+ BLKIO_STAT_SERVICED,
+ /* Total time spent waiting in scheduler queue in ns */
+ BLKIO_STAT_WAIT_TIME,
+ /* Number of IOs merged */
+ BLKIO_STAT_MERGED,
+ /* Number of IOs queued up */
+ BLKIO_STAT_QUEUED,
+ /* All the single valued stats go below this */
+ BLKIO_STAT_TIME,
+ BLKIO_STAT_SECTORS,
+#ifdef CONFIG_DEBUG_BLK_CGROUP
+ BLKIO_STAT_AVG_QUEUE_SIZE,
+ BLKIO_STAT_IDLE_TIME,
+ BLKIO_STAT_EMPTY_TIME,
+ BLKIO_STAT_GROUP_WAIT_TIME,
+ BLKIO_STAT_DEQUEUE
+#endif
+};
+
+enum stat_sub_type {
+ BLKIO_STAT_READ = 0,
+ BLKIO_STAT_WRITE,
+ BLKIO_STAT_SYNC,
+ BLKIO_STAT_ASYNC,
+ BLKIO_STAT_TOTAL
+};
+
+/* blkg state flags */
+enum blkg_state_flags {
```

```
+ BLKG_waiting = 0,
+ BLKG_idling,
+ BLKG_empty,
+};
+
+/* cgroup files owned by proportional weight policy */
+enum blkcg_file_name_prop {
+ BLKIO_PROP_weight = 1,
+ BLKIO_PROP_weight_device,
+ BLKIO_PROP_io_service_bytes,
+ BLKIO_PROP_io_serviced,
+ BLKIO_PROP_time,
+ BLKIO_PROP_sectors,
+ BLKIO_PROP_io_service_time,
+ BLKIO_PROP_io_wait_time,
+ BLKIO_PROP_io_merged,
+ BLKIO_PROP_io_queued,
+ BLKIO_PROP_avg_queue_size,
+ BLKIO_PROP_group_wait_time,
+ BLKIO_PROP_idle_time,
+ BLKIO_PROP_empty_time,
+ BLKIO_PROP_dequeue,
+};
+
+/* cgroup files owned by throttle policy */
+enum blkcg_file_name_throtl {
+ BLKIO_THROTL_read_bps_device,
+ BLKIO_THROTL_write_bps_device,
+ BLKIO_THROTL_read_iops_device,
+ BLKIO_THROTL_write_iops_device,
+ BLKIO_THROTL_io_service_bytes,
+ BLKIO_THROTL_io_serviced,
+};
+
+struct blkio_cgroup {
+ struct cgroup_subsys_state css;
+ unsigned int weight;
+ spinlock_t lock;
+ struct hlist_head blkg_list;
+ struct list_head policy_list; /* list of blkio_policy_node */
+};
+
+struct blkio_group_stats {
+ /* total disk time and nr sectors dispatched by this group */
+ uint64_t time;
+ uint64_t sectors;
+ uint64_t stat_arr[BLKIO_STAT_QUEUED + 1][BLKIO_STAT_TOTAL];
+#ifdef CONFIG_DEBUG_BLK_CGROUP
```

```
+ /* Sum of number of IOs queued across all samples */
+ uint64_t avg_queue_size_sum;
+ /* Count of samples taken for average */
+ uint64_t avg_queue_size_samples;
+ /* How many times this group has been removed from service tree */
+ unsigned long dequeue;
+
+ /* Total time spent waiting for it to be assigned a timeslice. */
+ uint64_t group_wait_time;
+ uint64_t start_group_wait_time;
+
+ /* Time spent idling for this blkio_group */
+ uint64_t idle_time;
+ uint64_t start_idle_time;
+ /*
+  * Total time when we have requests queued and do not contain the
+  * current active queue.
+  */
+ uint64_t empty_time;
+ uint64_t start_empty_time;
+ uint16_t flags;
+#endif
+};
+
+struct blkio_group {
+ /* An rcu protected unique identifier for the group */
+ void *key;
+ struct hlist_node blkcg_node;
+ unsigned short blkcg_id;
+ /* Store cgroup path */
+ char path[128];
+ /* The device MKDEV(major, minor), this group has been created for */
+ dev_t dev;
+ /* policy which owns this blk group */
+ enum blkio_policy_id plid;
+
+ /* Need to serialize the stats in the case of reset/update */
+ spinlock_t stats_lock;
+ struct blkio_group_stats stats;
+};
+
+struct blkio_policy_node {
+ struct list_head node;
+ dev_t dev;
+ /* This node belongs to max bw policy or porportional weight policy */
+ enum blkio_policy_id plid;
+ /* cgroup file to which this rule belongs to */
+ int fileid;
```

```
+
+ union {
+  unsigned int weight;
+  /*
+   * Rate read/write in terms of byptes per second
+   * Whether this rate represents read or write is determined
+   * by file type "fileid".
+   */
+  u64 bps;
+  unsigned int iops;
+ } val;
+};
+
+extern unsigned int blkcg_get_weight(struct blkio_cgroup *blkcg,
+         dev_t dev);
+extern uint64_t blkcg_get_read_bps(struct blkio_cgroup *blkcg,
+         dev_t dev);
+extern uint64_t blkcg_get_write_bps(struct blkio_cgroup *blkcg,
+         dev_t dev);
+extern unsigned int blkcg_get_read_iops(struct blkio_cgroup *blkcg,
+         dev_t dev);
+extern unsigned int blkcg_get_write_iops(struct blkio_cgroup *blkcg,
+         dev_t dev);
+
+typedef void (blkio_unlink_group_fn) (void *key, struct blkio_group *blkg);
+
+typedef void (blkio_update_group_weight_fn) (void *key,
+   struct blkio_group *blkg, unsigned int weight);
+typedef void (blkio_update_group_read_bps_fn) (void * key,
+   struct blkio_group *blkg, u64 read_bps);
+typedef void (blkio_update_group_write_bps_fn) (void *key,
+   struct blkio_group *blkg, u64 write_bps);
+typedef void (blkio_update_group_read_iops_fn) (void *key,
+   struct blkio_group *blkg, unsigned int read_iops);
+typedef void (blkio_update_group_write_iops_fn) (void *key,
+   struct blkio_group *blkg, unsigned int write_iops);
+
+struct blkio_policy_ops {
+ blkio_unlink_group_fn *blkio_unlink_group_fn;
+ blkio_update_group_weight_fn *blkio_update_group_weight_fn;
+ blkio_update_group_read_bps_fn *blkio_update_group_read_bps_fn;
+ blkio_update_group_write_bps_fn *blkio_update_group_write_bps_fn;
+ blkio_update_group_read_iops_fn *blkio_update_group_read_iops_fn;
+ blkio_update_group_write_iops_fn *blkio_update_group_write_iops_fn;
+};
+
+struct blkio_policy_type {
+ struct list_head list;
```

```
+ struct blkio_policy_ops ops;
+ enum blkio_policy_id plid;
+};
+
+/* Blkio controller policy registration */
+extern void blkio_policy_register(struct blkio_policy_type *);
+extern void blkio_policy_unregister(struct blkio_policy_type *);
+
+static inline char *blkg_path(struct blkio_group *blkg)
+{
+ return blkg->path;
+}
+
+#else
+
+struct blkio_group {
+};
+
+struct blkio_policy_type {
+};
+
+static inline void blkio_policy_register(struct blkio_policy_type *blkiop) { }
+static inline void blkio_policy_unregister(struct blkio_policy_type *blkiop) { }
+
+static inline char *blkg_path(struct blkio_group *blkg) { return NULL; }
+
+#endif
+
+#define BLKIO_WEIGHT_MIN 100
+#define BLKIO_WEIGHT_MAX 1000
+#define BLKIO_WEIGHT_DEFAULT 500
+
+#ifdef CONFIG_DEBUG_BLK_CGROUP
+void blkiocg_update_avg_queue_size_stats(struct blkio_group *blkg);
+void blkiocg_update_dequeue_stats(struct blkio_group *blkg,
+    unsigned long dequeue);
+void blkiocg_update_set_idle_time_stats(struct blkio_group *blkg);
+void blkiocg_update_idle_time_stats(struct blkio_group *blkg);
+void blkiocg_set_start_empty_time(struct blkio_group *blkg);
+
+#define BLKG_FLAG_FNS(name)      \
+static inline void blkio_mark_blkg_##name(    \
+  struct blkio_group_stats *stats)   \
+{        \
+ stats->flags |= (1 << BLKG_##name);    \
+}        \
+static inline void blkio_clear_blkg_##name(    \
+  struct blkio_group_stats *stats)   \
```

```
+{       \
+ stats->flags &= ~(1 << BLKG_##name);    \
+}       \
+static inline int blkio_blkg_##name(struct blkio_group_stats *stats) \
+{       \
+ return (stats->flags & (1 << BLKG_##name)) != 0;  \
+}       \
+
+BLKG_FLAG_FNS(waiting)
+BLKG_FLAG_FNS(idling)
+BLKG_FLAG_FNS(empty)
+#undef BLKG_FLAG_FNS
+#else
+static inline void blkiocg_update_avg_queue_size_stats(
+      struct blkio_group *blkg) {}
+static inline void blkiocg_update_dequeue_stats(struct blkio_group *blkg,
+      unsigned long dequeue) {}
+static inline void blkiocg_update_set_idle_time_stats(struct blkio_group *blkg)
+{}
+static inline void blkiocg_update_idle_time_stats(struct blkio_group *blkg) {}
+static inline void blkiocg_set_start_empty_time(struct blkio_group *blkg) {}
+#endif
+
+#if defined(CONFIG_BLK_CGROUP) || defined(CONFIG_BLK_CGROUP_MODULE)
+extern struct blkio_cgroup blkio_root_cgroup;
+extern bool blkio_cgroup_disabled(void);
+extern struct blkio_cgroup *cgroup_to_blkio_cgroup(struct cgroup *cgroup);
+extern void blkiocg_add_blkio_group(struct blkio_cgroup *blkcg,
+ struct blkio_group *blkg, void *key, dev_t dev,
+ enum blkio_policy_id plid);
+extern int blkiocg_del_blkio_group(struct blkio_group *blkg);
+extern struct blkio_group *blkiocg_lookup_group(struct blkio_cgroup *blkcg,
+      void *key);
+void blkiocg_update_timeslice_used(struct blkio_group *blkg,
+     unsigned long time);
+void blkiocg_update_dispatch_stats(struct blkio_group *blkg, uint64_t bytes,
+      bool direction, bool sync);
+void blkiocg_update_completion_stats(struct blkio_group *blkg,
+ uint64_t start_time, uint64_t io_start_time, bool direction, bool sync);
+void blkiocg_update_io_merged_stats(struct blkio_group *blkg, bool direction,
+     bool sync);
+void blkiocg_update_io_add_stats(struct blkio_group *blkg,
+  struct blkio_group *curr_blkg, bool direction, bool sync);
+void blkiocg_update_io_remove_stats(struct blkio_group *blkg,
+     bool direction, bool sync);
+#else
+struct cgroup;
+static inline bool blkio_cgroup_disabled(void) { return true; }
```

```
+static inline struct blkio_cgroup *
+cgroup_to_blkio_cgroup(struct cgroup *cgroup) { return NULL; }
+
+static inline void blkiocg_add_blkio_group(struct blkio_cgroup *blkcg,
+  struct blkio_group *blkg, void *key, dev_t dev,
+  enum blkio_policy_id plid) {}
+
+static inline int
+blkiocg_del_blkio_group(struct blkio_group *blkg) { return 0; }
+
+static inline struct blkio_group *
+blkiocg_lookup_group(struct blkio_cgroup *blkcg, void *key) { return NULL; }
+static inline void blkiocg_update_timeslice_used(struct blkio_group *blkg,
+      unsigned long time) {}
+static inline void blkiocg_update_dispatch_stats(struct blkio_group *blkg,
+    uint64_t bytes, bool direction, bool sync) {}
+static inline void blkiocg_update_completion_stats(struct blkio_group *blkg,
+  uint64_t start_time, uint64_t io_start_time, bool direction,
+  bool sync) {}
+static inline void blkiocg_update_io_merged_stats(struct blkio_group *blkg,
+      bool direction, bool sync) {}
+static inline void blkiocg_update_io_add_stats(struct blkio_group *blkg,
+  struct blkio_group *curr_blkg, bool direction, bool sync) {}
+static inline void blkiocg_update_io_remove_stats(struct blkio_group *blkg,
+      bool direction, bool sync) {}
+#endif
+#endif /* _BLK_CGROUP_H */
--
1.7.1
```

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

---

Subject: [PATCH 2/5] blk-cgroup: introduce task_to_blkio_cgroup()
Posted by Andrea Righi on Tue, 22 Feb 2011 17:12:53 GMT
View Forum Message <> Reply to Message

Introduce a helper function to retrieve a blkio cgroup from a task.

Signed-off-by: Andrea Righi <arighi@develer.com>

```
 block/blk-cgroup.c        |    7 +++++++
 include/linux/blk-cgroup.h |    4 ++++
 2 files changed, 11 insertions(+), 0 deletions(-)
```

```
diff --git a/block/blk-cgroup.c b/block/blk-cgroup.c
index bf9d354..f283ae1 100644
--- a/block/blk-cgroup.c
+++ b/block/blk-cgroup.c
@@ -107,6 +107,13 @@ blkio_policy_search_node(const struct blkio_cgroup *blkcg, dev_t dev,
  return NULL;
 }

+struct blkio_cgroup *task_to_blkio_cgroup(struct task_struct *task)
+{
+ return container_of(task_subsys_state(task, blkio_subsys_id),
+     struct blkio_cgroup, css);
+}
+EXPORT_SYMBOL_GPL(task_to_blkio_cgroup);
+
 struct blkio_cgroup *cgroup_to_blkio_cgroup(struct cgroup *cgroup)
 {
  return container_of(cgroup_subsys_state(cgroup, blkio_subsys_id),
diff --git a/include/linux/blk-cgroup.h b/include/linux/blk-cgroup.h
index 5e48204..41b59db 100644
--- a/include/linux/blk-cgroup.h
+++ b/include/linux/blk-cgroup.h
@@ -287,6 +287,7 @@ static inline void blkiocg_set_start_empty_time(struct blkio_group *blkg)
{}
 extern struct blkio_cgroup blkio_root_cgroup;
 extern bool blkio_cgroup_disabled(void);
 extern struct blkio_cgroup *cgroup_to_blkio_cgroup(struct cgroup *cgroup);
+extern struct blkio_cgroup *task_to_blkio_cgroup(struct task_struct *task);
 extern void blkiocg_add_blkio_group(struct blkio_cgroup *blkcg,
  struct blkio_group *blkg, void *key, dev_t dev,
  enum blkio_policy_id plid);
@@ -311,6 +312,9 @@ static inline bool blkio_cgroup_disabled(void) { return true; }
 static inline struct blkio_cgroup *
 cgroup_to_blkio_cgroup(struct cgroup *cgroup) { return NULL; }

+static inline struct blkio_cgroup *
+task_to_blkio_cgroup(struct task_struct *task) { return NULL; }
+
 static inline void blkiocg_add_blkio_group(struct blkio_cgroup *blkcg,
   struct blkio_group *blkg, void *key, dev_t dev,
   enum blkio_policy_id plid) {}
--
1.7.1
```

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

Subject: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Andrea Righi on Tue, 22 Feb 2011 17:12:54 GMT
View Forum Message <> Reply to Message

The page_cgroup infrastructure, currently available only for the memory cgroup controller, can be used to store the owner of each page and opportunely track the writeback IO. This information is encoded in the upper 16-bits of the page_cgroup->flags.

A owner can be identified using a generic ID number and the following interfaces are provided to store a retrieve this information:

```
  unsigned long page_cgroup_get_owner(struct page *page);
  int page_cgroup_set_owner(struct page *page, unsigned long id);
  int page_cgroup_copy_owner(struct page *npage, struct page *opage);
```

The blkio.throttle controller can use the cgroup css_id() as the owner's ID number.

Signed-off-by: Andrea Righi <arighi@develer.com>

```
---
 block/Kconfig              |    2 +
 block/blk-cgroup.c         |    6 ++
 include/linux/memcontrol.h  |    6 ++
 include/linux/mmzone.h      |    4 +-
 include/linux/page_cgroup.h |   33 ++++++++++-
 init/Kconfig                |    4 +
 mm/Makefile                 |    3 +-
 mm/memcontrol.c             |    6 ++
 mm/page_cgroup.c            |  129 ++++++++++++++++++++++++++++++++++++++++----
 9 files changed, 176 insertions(+), 17 deletions(-)

diff --git a/block/Kconfig b/block/Kconfig
index 60be1e0..1351ea8 100644
--- a/block/Kconfig
+++ b/block/Kconfig
@@ -80,6 +80,8 @@ config BLK_DEV_INTEGRITY
 config BLK_DEV_THROTTLING
  bool "Block layer bio throttling support"
  depends on BLK_CGROUP=y && EXPERIMENTAL
+ select MM_OWNER
+ select PAGE_TRACKING
  default n
  ---help---
  Block layer bio throttling support. It can be used to limit
diff --git a/block/blk-cgroup.c b/block/blk-cgroup.c
index f283ae1..5c57f0a 100644
--- a/block/blk-cgroup.c
+++ b/block/blk-cgroup.c
```

```
@@ -107,6 +107,12 @@ blkio_policy_search_node(const struct blkio_cgroup *blkcg, dev_t dev,
  return NULL;
 }

+bool blkio_cgroup_disabled(void)
+{
+ return blkio_subsys.disabled ? true : false;
+}
+EXPORT_SYMBOL_GPL(blkio_cgroup_disabled);
+
 struct blkio_cgroup *task_to_blkio_cgroup(struct task_struct *task)
 {
  return container_of(task_subsys_state(task, blkio_subsys_id),
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index f512e18..a8a7cf0 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -49,6 +49,8 @@ extern unsigned long mem_cgroup_isolate_pages(unsigned long
nr_to_scan,
  * (Of course, if memcg does memory allocation in future, GFP_KERNEL is sane.)
  */

+extern void __init_mem_page_cgroup(struct page_cgroup *pc);
+
 extern int mem_cgroup_newpage_charge(struct page *page, struct mm_struct *mm,
     gfp_t gfp_mask);
 /* for swap handling */
@@ -153,6 +155,10 @@ void mem_cgroup_split_huge_fixup(struct page *head, struct page
*tail);
 #else /* CONFIG_CGROUP_MEM_RES_CTLR */
 struct mem_cgroup;

+static inline void __init_mem_page_cgroup(struct page_cgroup *pc)
+{
+}
+
 static inline int mem_cgroup_newpage_charge(struct page *page,
      struct mm_struct *mm, gfp_t gfp_mask)
 {
diff --git a/include/linux/mmzone.h b/include/linux/mmzone.h
index 02ecb01..30a5938 100644
--- a/include/linux/mmzone.h
+++ b/include/linux/mmzone.h
@@ -615,7 +615,7 @@ typedef struct pglist_data {
  int nr_zones;
 #ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
  struct page *node_mem_map;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
```

```
+#ifdef CONFIG_PAGE_TRACKING
  struct page_cgroup *node_page_cgroup;
 #endif
 #endif
@@ -975,7 +975,7 @@ struct mem_section {

  /* See declaration of similar field in struct zone */
  unsigned long *pageblock_flags;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_PAGE_TRACKING
  /*
   * If !SPARSEMEM, pgdat doesn't have page_cgroup pointer. We use
   * section. (see memcontrol.h/page_cgroup.h about this.)
diff --git a/include/linux/page_cgroup.h b/include/linux/page_cgroup.h
index 6d6cb7a..cdd7728 100644
--- a/include/linux/page_cgroup.h
+++ b/include/linux/page_cgroup.h
@@ -1,7 +1,7 @@
 #ifndef __LINUX_PAGE_CGROUP_H
 #define __LINUX_PAGE_CGROUP_H

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_PAGE_TRACKING
 #include <linux/bit_spinlock.h>
 /*
  * Page Cgroup can be considered as an extended mem_map.
@@ -12,11 +12,38 @@
  */
 struct page_cgroup {
  unsigned long flags;
- struct mem_cgroup *mem_cgroup;
  struct page *page;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct mem_cgroup *mem_cgroup;
  struct list_head lru;  /* per cgroup LRU list */
+#endif
 };

+/*
+ * use lower 16 bits for flags and reserve the rest for the page tracking id
+ */
+#define PAGE_TRACKING_ID_SHIFT (16)
+#define PAGE_TRACKING_ID_BITS \
+  (8 * sizeof(unsigned long) - PAGE_TRACKING_ID_SHIFT)
+
+/* NOTE: must be called with page_cgroup() lock held */
+static inline unsigned long page_cgroup_get_id(struct page_cgroup *pc)
+{
```

```
+ return pc->flags >> PAGE_TRACKING_ID_SHIFT;
+}
+
+/* NOTE: must be called with page_cgroup() lock held */
+static inline void page_cgroup_set_id(struct page_cgroup *pc, unsigned long id)
+{
+ WARN_ON(id >= (1UL << PAGE_TRACKING_ID_BITS));
+ pc->flags &= (1UL << PAGE_TRACKING_ID_SHIFT) - 1;
+ pc->flags |= (unsigned long)(id << PAGE_TRACKING_ID_SHIFT);
+}
+
+unsigned long page_cgroup_get_owner(struct page *page);
+int page_cgroup_set_owner(struct page *page, unsigned long id);
+int page_cgroup_copy_owner(struct page *npage, struct page *opage);
+
 void __meminit pgdat_page_cgroup_init(struct pglist_data *pgdat);

 #ifdef CONFIG_SPARSEMEM
@@ -132,7 +159,7 @@ static inline void move_unlock_page_cgroup(struct page_cgroup *pc,
  local_irq_restore(*flags);
 }

-#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+#else /* CONFIG_PAGE_TRACKING */
 struct page_cgroup;

 static inline void __meminit pgdat_page_cgroup_init(struct pglist_data *pgdat)
diff --git a/init/Kconfig b/init/Kconfig
index be788c0..712a00a 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -633,6 +633,7 @@ config CGROUP_MEM_RES_CTLR
  bool "Memory Resource Controller for Control Groups"
  depends on RESOURCE_COUNTERS
  select MM_OWNER
+ select PAGE_TRACKING
  help
    Provides a memory resource controller that manages both anonymous
    memory and page cache. (See Documentation/cgroups/memory.txt)
@@ -813,6 +814,9 @@ config SCHED_AUTOGROUP
 config MM_OWNER
  bool

+config PAGE_TRACKING
+ bool
+
 config SYSFS_DEPRECATED
  bool "enable deprecated sysfs features to support old userspace tools"
```

```
  depends on SYSFS
diff --git a/mm/Makefile b/mm/Makefile
index 2b1b575..85448cc 100644
--- a/mm/Makefile
+++ b/mm/Makefile
@@ -38,7 +38,8 @@ obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
 obj-$(CONFIG_TRANSPARENT_HUGEPAGE) += huge_memory.o
-obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o page_cgroup.o
+obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o
+obj-$(CONFIG_PAGE_TRACKING) += page_cgroup.o
 obj-$(CONFIG_MEMORY_FAILURE) += memory-failure.o
 obj-$(CONFIG_HWPOISON_INJECT) += hwpoison-inject.o
 obj-$(CONFIG_DEBUG_KMEMLEAK) += kmemleak.o
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index da53a25..1f72c2b 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -5056,6 +5056,12 @@ struct cgroup_subsys mem_cgroup_subsys = {
  .use_id = 1,
 };

+void __meminit __init_mem_page_cgroup(struct page_cgroup *pc)
+{
+ pc->mem_cgroup = NULL;
+ INIT_LIST_HEAD(&pc->lru);
+}
+
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
 static int __init enable_swap_account(char *s)
 {
diff --git a/mm/page_cgroup.c b/mm/page_cgroup.c
index 5bffada..79214ae 100644
--- a/mm/page_cgroup.c
+++ b/mm/page_cgroup.c
@@ -2,6 +2,7 @@
 #include <linux/mmzone.h>
 #include <linux/bootmem.h>
 #include <linux/bit_spinlock.h>
+#include <linux/blk-cgroup.h>
 #include <linux/page_cgroup.h>
 #include <linux/hash.h>
 #include <linux/slab.h>
@@ -15,9 +16,8 @@ static void __meminit
 __init_page_cgroup(struct page_cgroup *pc, unsigned long pfn)
 {
  pc->flags = 0;
```

```
- pc->mem_cgroup = NULL;
  pc->page = pfn_to_page(pfn);
- INIT_LIST_HEAD(&pc->lru);
+ __init_mem_page_cgroup(pc);
 }
 static unsigned long total_usage;

@@ -75,7 +75,7 @@ void __init page_cgroup_init_flatmem(void)

  int nid, fail;

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

  for_each_online_node(nid)  {
@@ -84,12 +84,13 @@ void __init page_cgroup_init_flatmem(void)
    goto fail;
  }
  printk(KERN_INFO "allocated %ld bytes of page_cgroup\n", total_usage);
- printk(KERN_INFO "please try 'cgroup_disable=memory' option if you"
- " don't want memory cgroups\n");
+ printk(KERN_INFO
+  "try cgroup_disable=memory,blkio option if you don't want\n");
  return;
 fail:
  printk(KERN_CRIT "allocation of page_cgroup failed.\n");
- printk(KERN_CRIT "please try 'cgroup_disable=memory' boot option\n");
+ printk(KERN_CRIT
+  "try cgroup_disable=memory,blkio boot option\n");
  panic("Out of memory");
 }

@@ -258,7 +259,7 @@ void __init page_cgroup_init(void)
  unsigned long pfn;
  int fail = 0;

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

  for (pfn = 0; !fail && pfn < max_pfn; pfn += PAGES_PER_SECTION) {
@@ -267,14 +268,15 @@ void __init page_cgroup_init(void)
   fail = init_section_page_cgroup(pfn);
  }
  if (fail) {
-  printk(KERN_CRIT "try 'cgroup_disable=memory' boot option\n");
+  printk(KERN_CRIT
```

```
+   "try cgroup_disable=memory,blkio boot option\n");
   panic("Out of memory");
  } else {
   hotplug_memory_notifier(page_cgroup_callback, 0);
  }
  printk(KERN_INFO "allocated %ld bytes of page_cgroup\n", total_usage);
- printk(KERN_INFO "please try 'cgroup_disable=memory' option if you don't"
- " want memory cgroups\n");
+ printk(KERN_INFO
+  "try cgroup_disable=memory,blkio option if you don't want\n");
 }

 void __meminit pgdat_page_cgroup_init(struct pglist_data *pgdat)
@@ -282,8 +284,113 @@ void __meminit pgdat_page_cgroup_init(struct pglist_data *pgdat)
  return;
 }

-#endif
+#endif /* !defined(CONFIG_SPARSEMEM) */
+
+/**
+ * page_cgroup_get_owner() - get the owner ID of a page
+ * @page: the page we want to find the owner
+ *
+ * Returns the owner ID of the page, 0 means that the owner cannot be
+ * retrieved.
+ **/
+unsigned long page_cgroup_get_owner(struct page *page)
+{
+ struct page_cgroup *pc;
+ unsigned long ret;
+
+ pc = lookup_page_cgroup(page);
+ if (unlikely(!pc))
+  return 0;
+
+ lock_page_cgroup(pc);
+ ret = page_cgroup_get_id(pc);
+ unlock_page_cgroup(pc);
+ return ret;
+}
+
+/**
+ * page_cgroup_set_owner() - set the owner ID of a page
+ * @page: the page we want to tag
+ * @id:  the ID number that will be associated to page
+ *
+ * Returns 0 if the owner is correctly associated to the page. Returns a
```

```
+ * negative value in case of failure.
+ **/
+int page_cgroup_set_owner(struct page *page, unsigned long id)
+{
+ struct page_cgroup *pc;
+
+ pc = lookup_page_cgroup(page);
+ if (unlikely(!pc))
+  return -ENOENT;
+
+ lock_page_cgroup(pc);
+ page_cgroup_set_id(pc, id);
+ unlock_page_cgroup(pc);
+ return 0;
+}
+
+/*
+ * double_page_cgroup_lock() - safely lock two page cgroups
+ *
+ * The double_page_cgroup_lock() code uses the address of the page cgroup to be
+ * sure to acquire the locks always in the same order and avoid AB-BA deadlock.
+ */
+static void
+double_page_cgroup_lock(struct page_cgroup *pc1, struct page_cgroup *pc2)
+{
+ if (pc1 == pc2) {
+  lock_page_cgroup(pc1);
+ } else {
+  if (pc1 < pc2) {
+   lock_page_cgroup(pc1);
+   lock_page_cgroup(pc2);
+  } else {
+   lock_page_cgroup(pc2);
+   lock_page_cgroup(pc1);
+  }
+ }
+}

+/*
+ * double_unlock_page_cgroup() - safely unlock two page cgroups
+ */
+static void
+double_page_cgroup_unlock(struct page_cgroup *pc1, struct page_cgroup *pc2)
+{
+ unlock_page_cgroup(pc1);
+ if (pc1 != pc2)
+  unlock_page_cgroup(pc2);
+}
```

```
+
+/**
+ * page_cgroup_copy_owner() - copy the owner ID of a page into another page
+ * @npage: the page where we want to copy the owner
+ * @opage: the page from which we want to copy the ID
+ *
+ * Returns 0 if the owner is correctly associated to npage. Returns a negative
+ * value in case of failure.
+ **/
+int page_cgroup_copy_owner(struct page *npage, struct page *opage)
+{
+ struct page_cgroup *npc, *opc;
+ int id;
+
+ npc = lookup_page_cgroup(npage);
+ if (unlikely(!npc))
+  return -ENOENT;
+ opc = lookup_page_cgroup(opage);
+ if (unlikely(!opc))
+  return -ENOENT;
+ if (npc == opc)
+  return 0;
+ double_page_cgroup_lock(npc, opc);
+ id = page_cgroup_get_id(opc);
+ page_cgroup_set_id(npc, id);
+ double_page_cgroup_unlock(npc, opc);
+
+ return 0;
+}

 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP

--
1.7.1
```

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

---

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

Subject: [PATCH 5/5] blk-throttle: buffered and anonymous page tracking instrumentation
Posted by Andrea Righi on Tue, 22 Feb 2011 17:12:56 GMT
View Forum Message <> Reply to Message

Apply the buffered and anonymous page tracking hooks to the opportune
kernel functions.

Signed-off-by: Ryo Tsuruta <ryov@valinux.co.jp>
Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>
Signed-off-by: Andrea Righi <arighi@develer.com>
---
 fs/buffer.c | 1 +
 mm/bounce.c | 1 +
 mm/filemap.c | 1 +
 mm/memory.c | 5 +++++
 mm/page-writeback.c | 1 +
 mm/swap_state.c | 2 ++
 6 files changed, 11 insertions(+), 0 deletions(-)

```
diff --git a/fs/buffer.c b/fs/buffer.c
index 2219a76..6e473e6 100644
--- a/fs/buffer.c
+++ b/fs/buffer.c
@@ -667,6 +667,7 @@ static void __set_page_dirty(struct page *page,
  if (page->mapping) { /* Race with truncate? */
   WARN_ON_ONCE(warn && !PageUptodate(page));
   account_page_dirtied(page, mapping);
+  blk_throtl_set_filepage_owner(page, current->mm);
   radix_tree_tag_set(&mapping->page_tree,
     page_index(page), PAGECACHE_TAG_DIRTY);
  }
diff --git a/mm/bounce.c b/mm/bounce.c
index 1481de6..f85cafa 100644
--- a/mm/bounce.c
+++ b/mm/bounce.c
@@ -211,6 +211,7 @@ static void __blk_queue_bounce(struct request_queue *q, struct bio
**bio_orig,
   to->bv_len = from->bv_len;
   to->bv_offset = from->bv_offset;
   inc_zone_page_state(to->bv_page, NR_BOUNCE);
+  blk_throtl_copy_page_owner(to->bv_page, page);

   if (rw == WRITE) {
    char *vto, *vfrom;
diff --git a/mm/filemap.c b/mm/filemap.c
index 83a45d3..7fca2b8 100644
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -407,6 +407,7 @@ int add_to_page_cache_locked(struct page *page, struct address_space
*mapping,
      gfp_mask & GFP_RECLAIM_MASK);
  if (error)
```

```
   goto out;
+ blk_throtl_set_filepage_owner(page, current->mm);

  error = radix_tree_preload(gfp_mask & ~__GFP_HIGHMEM);
  if (error == 0) {
diff --git a/mm/memory.c b/mm/memory.c
index 8e8c183..ad5906b 100644
--- a/mm/memory.c
+++ b/mm/memory.c
@@ -52,6 +52,7 @@
 #include <linux/init.h>
 #include <linux/writeback.h>
 #include <linux/memcontrol.h>
+#include <linux/blkdev.h>
 #include <linux/mmu_notifier.h>
 #include <linux/kallsyms.h>
 #include <linux/swapops.h>
@@ -2391,6 +2392,7 @@ gotten:
    */
   ptep_clear_flush(vma, address, page_table);
   page_add_new_anon_rmap(new_page, vma, address);
+  blk_throtl_set_anonpage_owner(new_page, mm);
   /*
    * We call the notify macro here because, when using secondary
    * mmu page tables (such as kvm shadow page tables), we want the
@@ -2826,6 +2828,7 @@ static int do_swap_page(struct mm_struct *mm, struct vm_area_struct
*vma,
  flush_icache_page(vma, page);
  set_pte_at(mm, address, page_table, pte);
  do_page_add_anon_rmap(page, vma, address, exclusive);
+ blk_throtl_set_anonpage_owner(page, mm);
  /* It's better to call commit-charge after rmap is established */
  mem_cgroup_commit_charge_swapin(page, ptr);

@@ -2957,6 +2960,7 @@ static int do_anonymous_page(struct mm_struct *mm, struct
vm_area_struct *vma,

  inc_mm_counter_fast(mm, MM_ANONPAGES);
  page_add_new_anon_rmap(page, vma, address);
+ blk_throtl_set_anonpage_owner(page, mm);
 setpte:
  set_pte_at(mm, address, page_table, entry);

@@ -3106,6 +3110,7 @@ static int __do_fault(struct mm_struct *mm, struct vm_area_struct
*vma,
   if (anon) {
    inc_mm_counter_fast(mm, MM_ANONPAGES);
    page_add_new_anon_rmap(page, vma, address);
```

```
+   blk_throtl_set_anonpage_owner(page, mm);
   } else {
    inc_mm_counter_fast(mm, MM_FILEPAGES);
    page_add_file_rmap(page);
diff --git a/mm/page-writeback.c b/mm/page-writeback.c
index 2cb01f6..277c323 100644
--- a/mm/page-writeback.c
+++ b/mm/page-writeback.c
@@ -1168,6 +1168,7 @@ int __set_page_dirty_nobuffers(struct page *page)
    BUG_ON(mapping2 != mapping);
    WARN_ON_ONCE(!PagePrivate(page) && !PageUptodate(page));
    account_page_dirtied(page, mapping);
+   blk_throtl_set_filepage_owner(page, current->mm);
    radix_tree_tag_set(&mapping->page_tree,
     page_index(page), PAGECACHE_TAG_DIRTY);
   }
diff --git a/mm/swap_state.c b/mm/swap_state.c
index 5c8cfab..bc3a138 100644
--- a/mm/swap_state.c
+++ b/mm/swap_state.c
@@ -16,6 +16,7 @@
 #include <linux/pagemap.h>
 #include <linux/buffer_head.h>
 #include <linux/backing-dev.h>
+#include <linux/blkdev.h>
 #include <linux/pagevec.h>
 #include <linux/migrate.h>
 #include <linux/page_cgroup.h>
@@ -330,6 +331,7 @@ struct page *read_swap_cache_async(swp_entry_t entry, gfp_t
gfp_mask,
   /* May fail (-ENOMEM) if radix-tree node allocation failed. */
   __set_page_locked(new_page);
   SetPageSwapBacked(new_page);
+  blk_throtl_set_anonpage_owner(new_page, current->mm);
   err = __add_to_swap_cache(new_page, entry);
   if (likely(!err)) {
    radix_tree_preload_end();
--
1.7.1

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

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages

Posted by Chad Talbott on Tue, 22 Feb 2011 18:42:41 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 9:12 AM, Andrea Righi <arighi@develer.com> wrote:
> Add the tracking of buffered (writeback) and anonymous pages.
...
> ---
> 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
...
> +int blk_throtl_set_anonpage_owner(struct page *page, struct mm_struct *mm)
> +int blk_throtl_set_filepage_owner(struct page *page, struct mm_struct *mm)
> +int blk_throtl_copy_page_owner(struct page *npage, struct page *opage)

It would be nice if these were named blk_cgroup_*. This is arguably
more correct as the id comes from the blkio subsystem, and isn't
specific to blk-throttle. This will be more important very shortly,
as CFQ will be using this same cgroup id for async IO tracking soon.

is_kernel_io() is a good idea, it avoids a bug that we've run into
with CFQ async IO tracking. Why isn't PF_KTHREAD sufficient to cover
all kernel threads, including kswapd and those marked PF_MEMALLOC?

Thanks,
Chad

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

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Andrea Righi on Tue, 22 Feb 2011 19:12:47 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 10:42:41AM -0800, Chad Talbott wrote:
> On Tue, Feb 22, 2011 at 9:12 AM, Andrea Righi <arighi@develer.com> wrote:
> > Add the tracking of buffered (writeback) and anonymous pages.
> ...
> > ---
> > 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
> ...
> > +int blk_throtl_set_anonpage_owner(struct page *page, struct mm_struct *mm)
> > +int blk_throtl_set_filepage_owner(struct page *page, struct mm_struct *mm)
> > +int blk_throtl_copy_page_owner(struct page *npage, struct page *opage)
>
> It would be nice if these were named blk_cgroup_*. This is arguably
> more correct as the id comes from the blkio subsystem, and isn't
> specific to blk-throttle. This will be more important very shortly,
> as CFQ will be using this same cgroup id for async IO tracking soon.

Sounds reasonable. Will do in the next version.

>
> is_kernel_io() is a good idea, it avoids a bug that we've run into
> with CFQ async IO tracking. Why isn't PF_KTHREAD sufficient to cover
> all kernel threads, including kswapd and those marked PF_MEMALLOC?

With PF_MEMALLOC we're sure we don't add the page tracking overhead also
to non-kernel threads when memory gets low.

PF_KSWAPD is not probably needed, AFAICS it is only used by kswapd, that
is created by kthread_create() and so it has the PF_KTHREAD flag set.

Let's see if someone can give more deatils about that. In the while I'll
investigate and try to do some tests only with PF_KTHREAD.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Vivek Goyal on Tue, 22 Feb 2011 19:34:03 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 06:12:51PM +0100, Andrea Righi wrote:
> Currently the blkio.throttle controller only support synchronous IO requests.
> This means that we always look at the current task to identify the "owner" of
> each IO request.

>
> However dirty pages in the page cache can be wrote to disk 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 perspective.
>
> The same concept is also valid for anonymous pages involed in IO operations
> (swap).
>
> This patch allow to track the cgroup that originally dirtied each page in page
> cache and each anonymous page and pass these informations to the blk-throttle
> controller. These informations can be used to provide a better service level
> differentiation of buffered writes swap IO between different cgroups.
>

Hi Andrea,

Thanks for the patches. Before I look deeper into patches, had few
general queries/thoughts.

- So this requires memory controller to be enabled. Does it also require
  these to be co-mounted?

- Currently in throttling there is no limit on number of bios queued
  per group. I think this is not necessarily a very good idea because
  if throttling limits are low, we will build very long bio queues. So
  some AIO process can queue up lots of bios, consume lots of memory
  without getting blocked. I am sure there will be other side affects
  too. One of the side affects I noticed is that if an AIO process
  queues up too much of IO, and if I want to kill it now, it just hangs
  there for a really-2 long time (waiting for all the throttled IO
  to complete).

  So I was thinking of implementing either per group limit or per io
  context limit and after that process will be put to sleep. (something
  like request descriptor mechanism).

  If that's the case, then comes the question of what do to about kernel
  threads. Should they be blocked or not. If these are blocked then a
  fast group will also be indirectly throttled behind a slow group. If
  they are not then we still have the problem of too many bios queued
  in throttling layer.

- What to do about other kernel thread like kjournald which is doing
  IO on behalf of all the filesystem users. If data is also journalled
  then I think again everything got serialized and a faster group got
  backlogged behind a slower one.

- Two processes doing IO to same file and slower group will throttle
  IO for faster group also. (flushing is per inode).

I am not sure what are other common operations by kernel threads which
can make IO serialized.

Thanks
Vivek

> Testcase
> ========
> - create a cgroup with 1MiB/s write limit:
>   # mount -t cgroup -o blkio none /mnt/cgroup
>   # mkdir /mnt/cgroup/foo
>   # echo 8:0 $((1024 * 1024)) > /mnt/cgroup/foo/blkio.throttle.write_bps_device
>
> - move a task into the cgroup and run a dd to generate some writeback IO
>
> Results:
>   - 2.6.38-rc6 vanilla:
> $ cat /proc/$$/cgroup
>   1:blkio:/foo
> $ dd if=/dev/zero of=zero bs=1M count=1024 &
> $ dstat -df
>   --dsk/sda--
>   read writ
>   0   19M
>   0   19M
>   0   0
>   0   0
>   0   19M
>   ...
>
>   - 2.6.38-rc6 + blk-throttle writeback IO control:
> $ cat /proc/$$/cgroup
>   1:blkio:/foo
> $ dd if=/dev/zero of=zero bs=1M count=1024 &
> $ dstat -df
>   --dsk/sda--
>   read writ
>   0 1024
>   0 1024

>   0 1024
>   0 1024
>   0 1024
>   ...
>
> TODO
> ====
> - lots of testing
>
> Any feedback is welcome.
> -Andrea
>
> [PATCH 1/5] blk-cgroup: move blk-cgroup.h in include/linux/blk-cgroup.h
> [PATCH 2/5] blk-cgroup: introduce task_to_blkio_cgroup()
> [PATCH 3/5] page_cgroup: make page tracking available for blkio
> [PATCH 4/5] blk-throttle: track buffered and anonymous pages
> [PATCH 5/5] blk-throttle: buffered and anonymous page tracking instrumentation
>
> block/Kconfig | 2 +
> block/blk-cgroup.c | 15 ++-
> block/blk-cgroup.h | 335 --
> block/blk-throttle.c | 89 ++++++++++-
> block/cfq.h | 2 +-
> fs/buffer.c | 1 +
> include/linux/blk-cgroup.h | 341 ++
> include/linux/blkdev.h | 26 +++-
> include/linux/memcontrol.h | 6 +
> include/linux/mmzone.h | 4 +-
> include/linux/page_cgroup.h | 33 ++++-
> init/Kconfig | 4 +
> mm/Makefile | 3 +-
> mm/bounce.c | 1 +
> mm/filemap.c | 1 +
> mm/memcontrol.c | 6 +
> mm/memory.c | 5 +
> mm/page-writeback.c | 1 +
> mm/page_cgroup.c | 129 ++++++++++++++++--
> mm/swap_state.c | 2 +
> 20 files changed, 649 insertions(+), 357 deletions(-)

_____

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Jonathan Corbet on Tue, 22 Feb 2011 20:01:45 GMT

On Tue, 22 Feb 2011 18:12:54 +0100
Andrea Righi <arighi@develer.com> wrote:

> The page_cgroup infrastructure, currently available only for the memory
> cgroup controller, can be used to store the owner of each page and
> opportunely track the writeback IO. This information is encoded in
> the upper 16-bits of the page_cgroup->flags.
>
> A owner can be identified using a generic ID number and the following
> interfaces are provided to store a retrieve this information:
>
> unsigned long page_cgroup_get_owner(struct page *page);
> int page_cgroup_set_owner(struct page *page, unsigned long id);
> int page_cgroup_copy_owner(struct page *npage, struct page *opage);

My immediate observation is that you're not really tracking the "owner"
here - you're tracking an opaque 16-bit token known only to the block
controller in a field which - if changed by anybody other than the block
controller - will lead to mayhem in the block controller. I think it
might be clearer - and safer - to say "blkcg" or some such instead of
"owner" here.

I'm tempted to say it might be better to just add a pointer to your
throtl_grp structure into struct page_cgroup. Or maybe replace the
mem_cgroup pointer with a single pointer to struct css_set. Both of
those ideas, though, probably just add unwanted extra overhead now to gain
generality which may or may not be wanted in the future.

jon

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

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Vivek Goyal on Tue, 22 Feb 2011 20:49:28 GMT

On Tue, Feb 22, 2011 at 10:42:41AM -0800, Chad Talbott wrote:
> On Tue, Feb 22, 2011 at 9:12 AM, Andrea Righi <arighi@develer.com> wrote:
> > Add the tracking of buffered (writeback) and anonymous pages.
> ...
> > ---
> > 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
> ...
> > +int blk_throtl_set_anonpage_owner(struct page *page, struct mm_struct *mm)
> > +int blk_throtl_set_filepage_owner(struct page *page, struct mm_struct *mm)
> > +int blk_throtl_copy_page_owner(struct page *npage, struct page *opage)
>
> It would be nice if these were named blk_cgroup_*. This is arguably
> more correct as the id comes from the blkio subsystem, and isn't
> specific to blk-throttle. This will be more important very shortly,
> as CFQ will be using this same cgroup id for async IO tracking soon.

Should this really be all part of blk-cgroup.c and not blk-throttle.c
so that it can be used by CFQ code also down the line? Anyway all this
is not throttle specific as such but blkio controller specific.

Though function naming convetion is not great in blk-cgroup.c But
functions either have blkio_ prefix or blkiocg_ prefix.

Functions which are not directly dealing with cgroups or in general
are called by blk-throttle.c and/or cfq-iosched.c I have marked as
prefixed with "blkio_". Functions which directly deal with cgroup stuff
and register with cgroup subsystem for this controller are generally
having "blkiocg_" prefix.

In this case probably we can use probably blkio_ prefix.

Thanks
Vivek

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

---

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Vivek Goyal on Tue, 22 Feb 2011 21:00:30 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 06:12:55PM +0100, Andrea Righi wrote:
> 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);
```

Why are we exporting all these symbols?

Thanks
Vivek

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Vivek Goyal on Tue, 22 Feb 2011 21:22:53 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 06:12:54PM +0100, Andrea Righi wrote:

> The page_cgroup infrastructure, currently available only for the memory
> cgroup controller, can be used to store the owner of each page and
> opportunely track the writeback IO. This information is encoded in
> the upper 16-bits of the page_cgroup->flags.
>
> A owner can be identified using a generic ID number and the following
> interfaces are provided to store a retrieve this information:
>
> unsigned long page_cgroup_get_owner(struct page *page);
> int page_cgroup_set_owner(struct page *page, unsigned long id);
> int page_cgroup_copy_owner(struct page *npage, struct page *opage);
>
> The blkio.throttle controller can use the cgroup css_id() as the owner's
> ID number.
>
> Signed-off-by: Andrea Righi <arighi@develer.com>
> ---
> block/Kconfig | 2 +
> block/blk-cgroup.c | 6 ++
> include/linux/memcontrol.h | 6 ++
> include/linux/mmzone.h | 4 +-
> include/linux/page_cgroup.h | 33 ++++++++++-
> init/Kconfig | 4 +
> mm/Makefile | 3 +-
> mm/memcontrol.c | 6 ++
> mm/page_cgroup.c | 129 ++++++++++

> }
>
> +bool blkio_cgroup_disabled(void)
> +{
> + return blkio_subsys.disabled ? true : false;
> +}
> +EXPORT_SYMBOL_GPL(blkio_cgroup_disabled);
> +

I think there should be option to just disable this asyn feature of
blkio controller. So those who don't want it (running VMs with cache=none
option) and don't want to take the memory reservation hit should be
able to disable just ASYNC facility of blkio controller and not
the whole blkio controller facility.

Thanks
Vivek

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Vivek Goyal on Tue, 22 Feb 2011 21:57:20 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 01:01:45PM -0700, Jonathan Corbet wrote:
> On Tue, 22 Feb 2011 18:12:54 +0100
> Andrea Righi <arighi@develer.com> wrote:
>
> > The page_cgroup infrastructure, currently available only for the memory
> > cgroup controller, can be used to store the owner of each page and
> > opportunely track the writeback IO. This information is encoded in
> > the upper 16-bits of the page_cgroup->flags.
> >
> > A owner can be identified using a generic ID number and the following
> > interfaces are provided to store a retrieve this information:
> >
> > unsigned long page_cgroup_get_owner(struct page *page);
> > int page_cgroup_set_owner(struct page *page, unsigned long id);
> > int page_cgroup_copy_owner(struct page *npage, struct page *opage);
>
> My immediate observation is that you're not really tracking the "owner"
> here - you're tracking an opaque 16-bit token known only to the block
> controller in a field which - if changed by anybody other than the block
> controller - will lead to mayhem in the block controller. I think it
> might be clearer - and safer - to say "blkcg" or some such instead of

> "owner" here.
>
> I'm tempted to say it might be better to just add a pointer to your
> throtl_grp structure into struct page_cgroup.

throtl_grp might not even be present when page is being dirtied. When this IO is actually submitted to device, we migth end up creating new throtl_grp. I guess other concern here would be increasing the size of page_cgroup structure.

I guess you meant storing a pointer to blkio_cgroup, along the lines of storing a pointer to mem_cgroup. That also means extra 8 bytes and only one subsystem can use it at a time. So using upper bits of pc->flags is probably better.

> Or maybe replace the
> mem_cgroup pointer with a single pointer to struct css_set. Both of
> those ideas, though, probably just add unwanted extra overhead now to gain
> generality which may or may not be wanted in the future.

This sounds interesting. IIUC, then this single pointer will allow all the subsystems to use this single pointer to retireve respective cgroups without actually co-mounting them.

I am not sure how much work is involved in making it happen. Also not sure about the overhead involved in traversing one extra pointer. Also apart from blkio controller, have we practically felt the need of any other controller this info. (network controller?). Few days back we were experimenting with trying to control block IO bandwidth over NFS with the help of network controller but it did not really work well with host of issues and one them being losing the context information.

If storing css_set pointer is lot of work, may be for the time being we can go for this hardcoding that these bits are exclusively used by blkio controller and once some other controller wants to share it, then look for ways of how to do sharing.

Thanks
Vivek

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Andrea Righi on Tue, 22 Feb 2011 22:41:41 GMT

On Tue, Feb 22, 2011 at 02:34:03PM -0500, Vivek Goyal wrote:
> On Tue, Feb 22, 2011 at 06:12:51PM +0100, Andrea Righi wrote:
> > Currently the blkio.throttle controller only support synchronous IO requests.
> > This means that we always look at the current task to identify the "owner" of
> > each IO request.
> >
> > However dirty pages in the page cache can be wrote to disk 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 perspective.
> >
> > The same concept is also valid for anonymous pages involed in IO operations
> > (swap).
> >
> > This patch allow to track the cgroup that originally dirtied each page in page
> > cache and each anonymous page and pass these informations to the blk-throttle
> > controller. These informations can be used to provide a better service level
> > differentiation of buffered writes swap IO between different cgroups.
> >
>
> Hi Andrea,
>
> Thanks for the patches. Before I look deeper into patches, had few
> general queries/thoughts.
>
> - So this requires memory controller to be enabled. Does it also require
>   these to be co-mounted?

No and no. The blkio controller enables and uses the page_cgroup
functionality, but it doesn't depend on the memory controller. It
automatically selects CONFIG_MM_OWNER and CONFIG_PAGE_TRACKING (last
one added in PATCH 3/5) and this is sufficient to make page_cgroup
usable from any generic controller.

>
> - Currently in throttling there is no limit on number of bios queued
>   per group. I think this is not necessarily a very good idea because
>   if throttling limits are low, we will build very long bio queues. So
>   some AIO process can queue up lots of bios, consume lots of memory
>   without getting blocked. I am sure there will be other side affects
>   too. One of the side affects I noticed is that if an AIO process
>   queues up too much of IO, and if I want to kill it now, it just hangs

> there for a really-2 long time (waiting for all the throttled IO
> to complete).
>
> So I was thinking of implementing either per group limit or per io
> context limit and after that process will be put to sleep. (something
> like request descriptor mechanism).

io context limit seems a better solution for now. We can also expect some help from the memory controller, if we'll have the dirty memory limit per cgroup in the future the max amount of bios queued will be automatically limited by this functionality.

>
> If that's the case, then comes the question of what do to about kernel
> threads. Should they be blocked or not. If these are blocked then a
> fast group will also be indirectly throttled behind a slow group. If
> they are not then we still have the problem of too many bios queued
> in throttling layer.

I think kernel threads should be never forced to sleep, to avoid the classic "priority inversion" problem and create potential DoS in the system.

Also for this part the dirty memory limit per cgroup could help a lot, because a cgroup will never exceed its "quota" of dirty memory, so it will not be able to submit more than a certain amount of bios (corresponding to the dirty memory limit).

>
> - What to do about other kernel thread like kjournald which is doing
> IO on behalf of all the filesystem users. If data is also journalled
> then I think again everything got serialized and a faster group got
> backlogged behind a slower one.

This is the most critical issue IMHO.

The blkio controller should need some help from the filesystems to understand which IO request can be throttled and which cannot. At the moment critical IO requests (with critical I mean that are dependency for other requests) and non-critical requests are mixed together in a way that throttling a single request may stop a lot of other requests in the system, and at the block layer it's not possible to retrieve such informations.

I don't have a solution for this right now. Except looking at each filesystem implementation and try to understand how to pass these informations to the block layer.

>
> - Two processes doing IO to same file and slower group will throttle
>   IO for faster group also. (flushing is per inode).
>

I think we should accept to have an inode granularity. We could redesign
the writeback code to work per-cgroup / per-page, etc. but that would
add a huge overhead. The limit of inode granularity could be an
acceptable tradeoff, cgroups are supposed to work to different files
usually, well.. except when databases come into play (ouch!).

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Andrea Righi on Tue, 22 Feb 2011 23:01:47 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 01:01:45PM -0700, Jonathan Corbet wrote:
> On Tue, 22 Feb 2011 18:12:54 +0100
> Andrea Righi <arighi@develer.com> wrote:
>
> > The page_cgroup infrastructure, currently available only for the memory
> > cgroup controller, can be used to store the owner of each page and
> > opportunely track the writeback IO. This information is encoded in
> > the upper 16-bits of the page_cgroup->flags.
> >
> > A owner can be identified using a generic ID number and the following
> > interfaces are provided to store a retrieve this information:
> >
> >   unsigned long page_cgroup_get_owner(struct page *page);
> >   int page_cgroup_set_owner(struct page *page, unsigned long id);
> >   int page_cgroup_copy_owner(struct page *npage, struct page *opage);
>
> My immediate observation is that you're not really tracking the "owner"
> here - you're tracking an opaque 16-bit token known only to the block
> controller in a field which - if changed by anybody other than the block
> controller - will lead to mayhem in the block controller. I think it
> might be clearer - and safer - to say "blkcg" or some such instead of
> "owner" here.
>

Basically the idea here was to be as generic as possible and make this

feature potentially available also to other subsystems, so that cgroup
subsystems may represent whatever they want with the 16-bit token.
However, no more than a single subsystem may be able to use this feature
at the same time.

> I'm tempted to say it might be better to just add a pointer to your
> throtl_grp structure into struct page_cgroup. Or maybe replace the
> mem_cgroup pointer with a single pointer to struct css_set. Both of
> those ideas, though, probably just add unwanted extra overhead now to gain
> generality which may or may not be wanted in the future.

The pointer to css_set sounds good, but it would add additional space to
the page_cgroup struct. Now, page_cgroup is 40 bytes (in 64-bit arch)
and all of them are allocated at boot time. Using unused bits in
page_cgroup->flags is a choice with no overhead from this point of view.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Andrea Righi on Tue, 22 Feb 2011 23:03:54 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 03:49:28PM -0500, Vivek Goyal wrote:
> On Tue, Feb 22, 2011 at 10:42:41AM -0800, Chad Talbott wrote:
> > On Tue, Feb 22, 2011 at 9:12 AM, Andrea Righi <arighi@develer.com> wrote:
> > > Add the tracking of buffered (writeback) and anonymous pages.
> > ...
> > > ---
> > > 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
> > ...
> > > +int blk_throtl_set_anonpage_owner(struct page *page, struct mm_struct *mm)
> > > +int blk_throtl_set_filepage_owner(struct page *page, struct mm_struct *mm)
> > > +int blk_throtl_copy_page_owner(struct page *npage, struct page *opage)
> >

> > It would be nice if these were named blk_cgroup_*. This is arguably
> > more correct as the id comes from the blkio subsystem, and isn't
> > specific to blk-throttle. This will be more important very shortly,
> > as CFQ will be using this same cgroup id for async IO tracking soon.
>
> Should this really be all part of blk-cgroup.c and not blk-throttle.c
> so that it can be used by CFQ code also down the line? Anyway all this
> is not throttle specific as such but blkio controller specific.

Agreed.

>
> Though function naming convetion is not great in blk-cgroup.c But
> functions either have blkio_ prefix or blkiocg_ prefix.

ok.

>
> Functions which are not directly dealing with cgroups or in general
> are called by blk-throttle.c and/or cfq-iosched.c I have marked as
> prefixed with "blkio_". Functions which directly deal with cgroup stuff
> and register with cgroup subsystem for this controller are generally
> having "blkiocg_" prefix.
>
> In this case probably we can use probably blkio_ prefix.

ok.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Andrea Righi on Tue, 22 Feb 2011 23:05:34 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 04:00:30PM -0500, Vivek Goyal wrote:
> On Tue, Feb 22, 2011 at 06:12:55PM +0100, Andrea Righi wrote:
> > 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);
>
> Why are we exporting all these symbols?

Right. Probably a single one is enough:

 int blk_throtl_set_page_owner(struct page *page,
   struct mm_struct *mm, bool anon);

-Andrea

_____

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Vivek Goyal on Tue, 22 Feb 2011 23:06:30 GMT

On Wed, Feb 23, 2011 at 12:01:47AM +0100, Andrea Righi wrote:
> On Tue, Feb 22, 2011 at 01:01:45PM -0700, Jonathan Corbet wrote:
> > On Tue, 22 Feb 2011 18:12:54 +0100
> > Andrea Righi <arighi@develer.com> wrote:
> >
> > > The page_cgroup infrastructure, currently available only for the memory
> > > cgroup controller, can be used to store the owner of each page and
> > > opportunely track the writeback IO. This information is encoded in
> > > the upper 16-bits of the page_cgroup->flags.
> > >
> > > A owner can be identified using a generic ID number and the following
> > > interfaces are provided to store a retrieve this information:
> > >
> > > unsigned long page_cgroup_get_owner(struct page *page);
> > > int page_cgroup_set_owner(struct page *page, unsigned long id);
> > > int page_cgroup_copy_owner(struct page *npage, struct page *opage);
> >
> > My immediate observation is that you're not really tracking the "owner"
> > here - you're tracking an opaque 16-bit token known only to the block
> > controller in a field which - if changed by anybody other than the block
> > controller - will lead to mayhem in the block controller. I think it
> > might be clearer - and safer - to say "blkcg" or some such instead of
> > "owner" here.
> >
>
> Basically the idea here was to be as generic as possible and make this
> feature potentially available also to other subsystems, so that cgroup
> subsystems may represent whatever they want with the 16-bit token.
> However, no more than a single subsystem may be able to use this feature
> at the same time.
>
> > I'm tempted to say it might be better to just add a pointer to your
> > throtl_grp structure into struct page_cgroup. Or maybe replace the
> > mem_cgroup pointer with a single pointer to struct css_set. Both of
> > those ideas, though, probably just add unwanted extra overhead now to gain
> > generality which may or may not be wanted in the future.
>
> The pointer to css_set sounds good, but it would add additional space to
> the page_cgroup struct. Now, page_cgroup is 40 bytes (in 64-bit arch)
> and all of them are allocated at boot time. Using unused bits in
> page_cgroup->flags is a choice with no overhead from this point of view.

I think John suggested replacing mem_cgroup pointer with css_set so that
size of the strcuture does not increase but it leads extra level of
indirection.

Thanks

Vivek

_____

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Andrea Righi on Tue, 22 Feb 2011 23:08:40 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 04:22:53PM -0500, Vivek Goyal wrote:
> On Tue, Feb 22, 2011 at 06:12:54PM +0100, Andrea Righi wrote:
> > The page_cgroup infrastructure, currently available only for the memory
> > cgroup controller, can be used to store the owner of each page and
> > opportunely track the writeback IO. This information is encoded in
> > the upper 16-bits of the page_cgroup->flags.
> >
> > A owner can be identified using a generic ID number and the following
> > interfaces are provided to store a retrieve this information:
> >
> > unsigned long page_cgroup_get_owner(struct page *page);
> > int page_cgroup_set_owner(struct page *page, unsigned long id);
> > int page_cgroup_copy_owner(struct page *npage, struct page *opage);
> >
> > The blkio.throttle controller can use the cgroup css_id() as the owner's
> > ID number.
> >
> > Signed-off-by: Andrea Righi <arighi@develer.com>
> > ---
> > block/Kconfig | 2 +
> > block/blk-cgroup.c | 6 ++
> > include/linux/memcontrol.h | 6 ++
> > include/linux/mmzone.h | 4 +-
> > include/linux/page_cgroup.h | 33 ++++++++++-
> > init/Kconfig | 4 +
> > mm/Makefile | 3 +-
> > mm/memcontrol.c | 6 ++
> > mm/page_cgroup.c | 129 ++-
> > 9 files changed, 176 insertions(+), 17 deletions(-)
> >
> > diff --git a/block/Kconfig b/block/Kconfig
> > index 60be1e0..1351ea8 100644
> > --- a/block/Kconfig
> > +++ b/block/Kconfig
> > @@ -80,6 +80,8 @@ config BLK_DEV_INTEGRITY
> > config BLK_DEV_THROTTLING
> > bool "Block layer bio throttling support"

> >   depends on BLK_CGROUP=y && EXPERIMENTAL
> > + select MM_OWNER
> > + select PAGE_TRACKING
> >   default n
> >   ---help---
> >   Block layer bio throttling support. It can be used to limit
> > diff --git a/block/blk-cgroup.c b/block/blk-cgroup.c
> > index f283ae1..5c57f0a 100644
> > --- a/block/blk-cgroup.c
> > +++ b/block/blk-cgroup.c
> > @@ -107,6 +107,12 @@ blkio_policy_search_node(const struct blkio_cgroup *blkcg, dev_t
dev,
> >   return NULL;
> > }
> >
> > +bool blkio_cgroup_disabled(void)
> > +{
> > + return blkio_subsys.disabled ? true : false;
> > +}
> > +EXPORT_SYMBOL_GPL(blkio_cgroup_disabled);
> > +
>
> I think there should be option to just disable this asyn feature of
> blkio controller. So those who don't want it (running VMs with cache=none
> option) and don't want to take the memory reservation hit should be
> able to disable just ASYNC facility of blkio controller and not
> the whole blkio controller facility.

Definitely a better choice.

OK, I'll apply all your suggestions and post a new version of the patch.

Thanks for the review!
-Andrea

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Jonathan Corbet on Tue, 22 Feb 2011 23:21:47 GMT
View Forum Message <> Reply to Message

On Tue, 22 Feb 2011 18:06:30 -0500
Vivek Goyal <vgoyal@redhat.com> wrote:

> I think John suggested replacing mem_cgroup pointer with css_set so that

> size of the strcuture does not increase but it leads extra level of
> indirection.

That is what I was thinking. But I did also say it's probably premature
generalization at this point, especially given that there'd be a runtime
cost.

jon

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Jonathan Corbet on Tue, 22 Feb 2011 23:27:29 GMT
View Forum Message <> Reply to Message

On Wed, 23 Feb 2011 00:01:47 +0100
Andrea Righi <arighi@develer.com> wrote:

> > My immediate observation is that you're not really tracking the "owner"
> > here - you're tracking an opaque 16-bit token known only to the block
> > controller in a field which - if changed by anybody other than the block
> > controller - will lead to mayhem in the block controller. I think it
> > might be clearer - and safer - to say "blkcg" or some such instead of
> > "owner" here.
>
> Basically the idea here was to be as generic as possible and make this
> feature potentially available also to other subsystems, so that cgroup
> subsystems may represent whatever they want with the 16-bit token.
> However, no more than a single subsystem may be able to use this feature
> at the same time.

That makes me nervous; it can't really be used that way unless we want to
say that certain controllers are fundamentally incompatible and can't be
allowed to play together. For whatever my $0.02 are worth (given the
state of the US dollar, that's not a whole lot), I'd suggest keeping the
current mechanism, but make it clear that it belongs to your controller.
If and when another controller comes along with a need for similar
functionality, somebody can worry about making it more general.

jon

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Andrea Righi on Tue, 22 Feb 2011 23:37:18 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 06:06:30PM -0500, Vivek Goyal wrote:
> On Wed, Feb 23, 2011 at 12:01:47AM +0100, Andrea Righi wrote:
> > On Tue, Feb 22, 2011 at 01:01:45PM -0700, Jonathan Corbet wrote:
> > > On Tue, 22 Feb 2011 18:12:54 +0100
> > > Andrea Righi <arighi@develer.com> wrote:
> > >
> > > > The page_cgroup infrastructure, currently available only for the memory
> > > > cgroup controller, can be used to store the owner of each page and
> > > > opportunely track the writeback IO. This information is encoded in
> > > > the upper 16-bits of the page_cgroup->flags.
> > > >
> > > > A owner can be identified using a generic ID number and the following
> > > > interfaces are provided to store a retrieve this information:
> > > >
> > > > unsigned long page_cgroup_get_owner(struct page *page);
> > > > int page_cgroup_set_owner(struct page *page, unsigned long id);
> > > > int page_cgroup_copy_owner(struct page *npage, struct page *opage);
> > >
> > > My immediate observation is that you're not really tracking the "owner"
> > > here - you're tracking an opaque 16-bit token known only to the block
> > > controller in a field which - if changed by anybody other than the block
> > > controller - will lead to mayhem in the block controller. I think it
> > > might be clearer - and safer - to say "blkcg" or some such instead of
> > > "owner" here.
> > >
> >
> > Basically the idea here was to be as generic as possible and make this
> > feature potentially available also to other subsystems, so that cgroup
> > subsystems may represent whatever they want with the 16-bit token.
> > However, no more than a single subsystem may be able to use this feature
> > at the same time.
> >
> > > I'm tempted to say it might be better to just add a pointer to your
> > > throtl_grp structure into struct page_cgroup. Or maybe replace the
> > > mem_cgroup pointer with a single pointer to struct css_set. Both of
> > > those ideas, though, probably just add unwanted extra overhead now to gain
> > > generality which may or may not be wanted in the future.
> >
> > The pointer to css_set sounds good, but it would add additional space to
> > the page_cgroup struct. Now, page_cgroup is 40 bytes (in 64-bit arch)
> > and all of them are allocated at boot time. Using unused bits in
> > page_cgroup->flags is a choice with no overhead from this point of view.
>
> I think John suggested replacing mem_cgroup pointer with css_set so that
> size of the strcuture does not increase but it leads extra level of

> indirection.

OK, got it sorry.

So, IIUC we save css_set pointer and get a struct cgroup as following:

```
  struct cgroup *cgrp = css_set->subsys[subsys_id]->cgroup;
```

Then, for example to get the mem_cgroup reference:

```
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
```

It seems a lot of indirections, but I may have done something wrong or
there could be a simpler way to do it.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Andrea Righi on Tue, 22 Feb 2011 23:48:33 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 04:27:29PM -0700, Jonathan Corbet wrote:
> On Wed, 23 Feb 2011 00:01:47 +0100
> Andrea Righi <arighi@develer.com> wrote:
>
> > > My immediate observation is that you're not really tracking the "owner"
> > > here - you're tracking an opaque 16-bit token known only to the block
> > > controller in a field which - if changed by anybody other than the block
> > > controller - will lead to mayhem in the block controller. I think it
> > > might be clearer - and safer - to say "blkcg" or some such instead of
> > > "owner" here.
> >
> > Basically the idea here was to be as generic as possible and make this
> > feature potentially available also to other subsystems, so that cgroup
> > subsystems may represent whatever they want with the 16-bit token.
> > However, no more than a single subsystem may be able to use this feature
> > at the same time.
>
> That makes me nervous; it can't really be used that way unless we want to
> say that certain controllers are fundamentally incompatible and can't be
> allowed to play together. For whatever my $0.02 are worth (given the
> state of the US dollar, that's not a whole lot), I'd suggest keeping the

> current mechanism, but make it clear that it belongs to your controller.
> If and when another controller comes along with a need for similar
> functionality, somebody can worry about making it more general.

OK, I understand. I'll use "blkio" instead of "owner". Also because I wouldn't like to introduce additional logic and overhead to check if two controllers are using this feature at the same time. Better to hard-code this information in the name of the functions.

Probably the most generic solution is the one that you suggested: replace the mem_cgroup with a pointer to css_set. I'll also try to investigate this way.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Vivek Goyal on Wed, 23 Feb 2011 00:03:58 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 11:41:41PM +0100, Andrea Righi wrote:
> On Tue, Feb 22, 2011 at 02:34:03PM -0500, Vivek Goyal wrote:
> > On Tue, Feb 22, 2011 at 06:12:51PM +0100, Andrea Righi wrote:
> > > Currently the blkio.throttle controller only support synchronous IO requests.
> > > This means that we always look at the current task to identify the "owner" of
> > > each IO request.
> > >
> > > However dirty pages in the page cache can be wrote to disk 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 perspective.
> > >
> > > The same concept is also valid for anonymous pages involed in IO operations
> > > (swap).
> > >
> > > This patch allow to track the cgroup that originally dirtied each page in page
> > > cache and each anonymous page and pass these informations to the blk-throttle
> > > controller. These informations can be used to provide a better service level

> > > differentiation of buffered writes swap IO between different cgroups.
> > >
> >
> > Hi Andrea,
> >
> > Thanks for the patches. Before I look deeper into patches, had few
> > general queries/thoughts.
> >
> > - So this requires memory controller to be enabled. Does it also require
> >   these to be co-mounted?
>
> No and no. The blkio controller enables and uses the page_cgroup
> functionality, but it doesn't depend on the memory controller. It
> automatically selects CONFIG_MM_OWNER and CONFIG_PAGE_TRACKING (last
> one added in PATCH 3/5) and this is sufficient to make page_cgroup
> usable from any generic controller.
>
> >
> > - Currently in throttling there is no limit on number of bios queued
> >   per group. I think this is not necessarily a very good idea because
> >   if throttling limits are low, we will build very long bio queues. So
> >   some AIO process can queue up lots of bios, consume lots of memory
> >   without getting blocked. I am sure there will be other side affects
> >   too. One of the side affects I noticed is that if an AIO process
> >   queues up too much of IO, and if I want to kill it now, it just hangs
> >   there for a really-2 long time (waiting for all the throttled IO
> >   to complete).
> >
> >   So I was thinking of implementing either per group limit or per io
> >   context limit and after that process will be put to sleep. (something
> >   like request descriptor mechanism).
>
> io context limit seems a better solution for now. We can also expect
> some help from the memory controller, if we'll have the dirty memory
> limit per cgroup in the future the max amount of bios queued will be
> automatically limited by this functionality.
>
> >
> >   If that's the case, then comes the question of what do to about kernel
> >   threads. Should they be blocked or not. If these are blocked then a
> >   fast group will also be indirectly throttled behind a slow group. If
> >   they are not then we still have the problem of too many bios queued
> >   in throttling layer.
>
> I think kernel threads should be never forced to sleep, to avoid the
> classic "priority inversion" problem and create potential DoS in the
> system.
>

> Also for this part the dirty memory limit per cgroup could help a lot,
> because a cgroup will never exceed its "quota" of dirty memory, so it
> will not be able to submit more than a certain amount of bios
> (corresponding to the dirty memory limit).

Per memory cgroup dirty ratio should help a bit. But with intentional
throttling we always run the risk of faster groups getting stuck behind
slower groups.

Even in the case of buffered WRITES, are you able to run two buffered
WRITE streams in two groups and then throttle these to respective rates.
It might be interesting to run that and see what happens.

Practically I feel we shall have to run this with per cgroup memory
dirty ratio bit hence coumount with memory controller.

>
> >
> > - What to do about other kernel thread like kjournald which is doing
> > IO on behalf of all the filesystem users. If data is also journalled
> > then I think again everything got serialized and a faster group got
> > backlogged behind a slower one.
>
> This is the most critical issue IMHO.
>
> The blkio controller should need some help from the filesystems to
> understand which IO request can be throttled and which cannot. At the
> moment critical IO requests (with critical I mean that are dependency
> for other requests) and non-critical requests are mixed together in a
> way that throttling a single request may stop a lot of other requests in
> the system, and at the block layer it's not possible to retrieve such
> informations.
>
> I don't have a solution for this right now. Except looking at each
> filesystem implementation and try to understand how to pass these
> informations to the block layer.

True. This is very important issue which needs to be sorted out. Because
if due to journalling if file IO gets serialized behind a really slow
artificially throlled group, it might be very bad for the overall
filesystem performance.

I am CCing linux-fsdevel, and hopefully filesystem guys there can give
us some ideas regarding how it can be handeled.

>
> >
> > - Two processes doing IO to same file and slower group will throttle

> > IO for faster group also. (flushing is per inode).
> >
>
> I think we should accept to have an inode granularity. We could redesign
> the writeback code to work per-cgroup / per-page, etc. but that would
> add a huge overhead. The limit of inode granularity could be an
> acceptable tradeoff, cgroups are supposed to work to different files
> usually, well.. except when databases come into play (ouch!).

Agreed. Granularity of per inode level might be accetable in many
cases. Again, I am worried faster group getting stuck behind slower
group.

I am wondering if we are trying to solve the problem of ASYNC write throttling
at wrong layer. Should ASYNC IO be throttled before we allow task to write to
page cache. The way we throttle the process based on dirty ratio, can we
just check for throttle limits also there or something like that.(I think
that's what you had done in your initial throttling controller implementation?)

Thanks
Vivek

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

---

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Vivek Goyal on Wed, 23 Feb 2011 00:07:19 GMT
View Forum Message <> Reply to Message

On Wed, Feb 23, 2011 at 12:05:34AM +0100, Andrea Righi wrote:
> On Tue, Feb 22, 2011 at 04:00:30PM -0500, Vivek Goyal wrote:
> > On Tue, Feb 22, 2011 at 06:12:55PM +0100, Andrea Righi wrote:
> > > 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);
> >
> > Why are we exporting all these symbols?
>
> Right. Probably a single one is enough:
>
> int blk_throtl_set_page_owner(struct page *page,
>   struct mm_struct *mm, bool anon);

Who is going to use this single export? Which module?

Thanks
Vivek

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by KAMEZAWA Hiroyuki on Wed, 23 Feb 2011 04:49:10 GMT
View Forum Message <> Reply to Message

On Wed, 23 Feb 2011 00:37:18 +0100
Andrea Righi <arighi@develer.com> wrote:

> On Tue, Feb 22, 2011 at 06:06:30PM -0500, Vivek Goyal wrote:

> > On Wed, Feb 23, 2011 at 12:01:47AM +0100, Andrea Righi wrote:
> > > On Tue, Feb 22, 2011 at 01:01:45PM -0700, Jonathan Corbet wrote:
> > > > On Tue, 22 Feb 2011 18:12:54 +0100
> > > > Andrea Righi <arighi@develer.com> wrote:
> > > >
> > > > > The page_cgroup infrastructure, currently available only for the memory
> > > > > cgroup controller, can be used to store the owner of each page and
> > > > > opportunely track the writeback IO. This information is encoded in
> > > > > the upper 16-bits of the page_cgroup->flags.
> > > > >
> > > > > A owner can be identified using a generic ID number and the following
> > > > > interfaces are provided to store a retrieve this information:
> > > > >
> > > > >   unsigned long page_cgroup_get_owner(struct page *page);
> > > > >   int page_cgroup_set_owner(struct page *page, unsigned long id);
> > > > >   int page_cgroup_copy_owner(struct page *npage, struct page *opage);
> > > >
> > > > My immediate observation is that you're not really tracking the "owner"
> > > > here - you're tracking an opaque 16-bit token known only to the block
> > > > controller in a field which - if changed by anybody other than the block
> > > > controller - will lead to mayhem in the block controller. I think it
> > > > might be clearer - and safer - to say "blkcg" or some such instead of
> > > > "owner" here.
> > > >
> > >
> > > Basically the idea here was to be as generic as possible and make this
> > > feature potentially available also to other subsystems, so that cgroup
> > > subsystems may represent whatever they want with the 16-bit token.
> > > However, no more than a single subsystem may be able to use this feature
> > > at the same time.
> > >
> > > > I'm tempted to say it might be better to just add a pointer to your
> > > > throtl_grp structure into struct page_cgroup. Or maybe replace the
> > > > mem_cgroup pointer with a single pointer to struct css_set. Both of
> > > > those ideas, though, probably just add unwanted extra overhead now to gain
> > > > generality which may or may not be wanted in the future.
> > >
> > > The pointer to css_set sounds good, but it would add additional space to
> > > the page_cgroup struct. Now, page_cgroup is 40 bytes (in 64-bit arch)
> > > and all of them are allocated at boot time. Using unused bits in
> > > page_cgroup->flags is a choice with no overhead from this point of view.
> >
> > I think John suggested replacing mem_cgroup pointer with css_set so that
> > size of the strcuture does not increase but it leads extra level of
> > indirection.
>
> OK, got it sorry.
>

> So, IIUC we save css_set pointer and get a struct cgroup as following:
>
> struct cgroup *cgrp = css_set->subsys[subsys_id]->cgroup;
>
> Then, for example to get the mem_cgroup reference:
>
> struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
>
> It seems a lot of indirections, but I may have done something wrong or
> there could be a simpler way to do it.
>

Then, page_cgroup should have reference count on css_set and make tons of
atomic ops.

BTW, bits of pc->flags are used for storing sectionID or nodeID.
Please clarify your 16bit never breaks that information. And please keep
more 4-5 flags for dirty_ratio support of memcg.

I wonder I can make pc->mem_cgroup to be pc->memid(16bit), then,
==
```
static inline struct mem_cgroup *get_memcg_from_pc(struct page_cgroup *pc)
{
    struct cgroup_subsys_state *css = css_lookup(&mem_cgroup_subsys, pc->memid);
    return container_of(css, struct mem_cgroup, css);
}
```
==
Overhead will be seen at updating file statistics and LRU management.

But, hmm, can't you do that tracking without page_cgroup ?
Because the number of dirty/writeback pages are far smaller than total pages,
chasing I/O with dynamic structure is not very bad..

prepareing [pfn -> blkio] record table and move that information to struct bio
in dynamic way is very difficult ?

Thanks,
-Kame

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

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control

Posted by Andrea Righi on Wed, 23 Feb 2011 08:32:06 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 07:03:58PM -0500, Vivek Goyal wrote:
> > I think we should accept to have an inode granularity. We could redesign
> > the writeback code to work per-cgroup / per-page, etc. but that would
> > add a huge overhead. The limit of inode granularity could be an
> > acceptable tradeoff, cgroups are supposed to work to different files
> > usually, well.. except when databases come into play (ouch!).
>
> Agreed. Granularity of per inode level might be accetable in many
> cases. Again, I am worried faster group getting stuck behind slower
> group.
>
> I am wondering if we are trying to solve the problem of ASYNC write throttling
> at wrong layer. Should ASYNC IO be throttled before we allow task to write to
> page cache. The way we throttle the process based on dirty ratio, can we
> just check for throttle limits also there or something like that.(I think
> that's what you had done in your initial throttling controller implementation?)

Right. This is exactly the same approach I've used in my old throttling controller: throttle sync READs and WRITEs at the block layer and async WRITEs when the task is dirtying memory pages.

This is probably the simplest way to resolve the problem of faster group getting blocked by slower group, but the controller will be a little bit more leaky, because the writeback IO will be never throttled and we'll see some limited IO spikes during the writeback. However, this is always a better solution IMHO respect to the current implementation that is affected by that kind of priority inversion problem.

I can try to add this logic to the current blk-throttle controller if you think it is worth to test it.

-Andrea

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

Subject: Re: [PATCH 4/5] blk-throttle: track buffered and anonymous pages
Posted by Andrea Righi on Wed, 23 Feb 2011 08:37:40 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 07:07:19PM -0500, Vivek Goyal wrote:
> On Wed, Feb 23, 2011 at 12:05:34AM +0100, Andrea Righi wrote:
> > On Tue, Feb 22, 2011 at 04:00:30PM -0500, Vivek Goyal wrote:

> > > On Tue, Feb 22, 2011 at 06:12:55PM +0100, Andrea Righi wrote:
> > > > 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);
> > >
> > > Why are we exporting all these symbols?
> >
> > Right. Probably a single one is enough:
> >
> > int blk_throtl_set_page_owner(struct page *page,
> > struct mm_struct *mm, bool anon);
>
> Who is going to use this single export? Which module?
>

I was actually thinking at some filesystem modules, but I was wrong,
because at the moment no one needs the export. I'll remove it in the
next version of the patch.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by Andrea Righi on Wed, 23 Feb 2011 08:59:11 GMT
View Forum Message <> Reply to Message

On Wed, Feb 23, 2011 at 01:49:10PM +0900, KAMEZAWA Hiroyuki wrote:
> On Wed, 23 Feb 2011 00:37:18 +0100
> Andrea Righi <arighi@develer.com> wrote:
>
> > On Tue, Feb 22, 2011 at 06:06:30PM -0500, Vivek Goyal wrote:
> > > On Wed, Feb 23, 2011 at 12:01:47AM +0100, Andrea Righi wrote:
> > > > On Tue, Feb 22, 2011 at 01:01:45PM -0700, Jonathan Corbet wrote:
> > > > > On Tue, 22 Feb 2011 18:12:54 +0100
> > > > > Andrea Righi <arighi@develer.com> wrote:
> > > > >
> > > > > > The page_cgroup infrastructure, currently available only for the memory
> > > > > > cgroup controller, can be used to store the owner of each page and
> > > > > > opportunely track the writeback IO. This information is encoded in
> > > > > > the upper 16-bits of the page_cgroup->flags.
> > > > > >
> > > > > > A owner can be identified using a generic ID number and the following
> > > > > > interfaces are provided to store a retrieve this information:
> > > > > >
> > > > > > unsigned long page_cgroup_get_owner(struct page *page);
> > > > > > int page_cgroup_set_owner(struct page *page, unsigned long id);
> > > > > > int page_cgroup_copy_owner(struct page *npage, struct page *opage);
> > > > >
> > > > > My immediate observation is that you're not really tracking the "owner"
> > > > > here - you're tracking an opaque 16-bit token known only to the block
> > > > > controller in a field which - if changed by anybody other than the block
> > > > > controller - will lead to mayhem in the block controller. I think it
> > > > > might be clearer - and safer - to say "blkcg" or some such instead of
> > > > > "owner" here.
> > > > >
> > > >
> > > > Basically the idea here was to be as generic as possible and make this
> > > > feature potentially available also to other subsystems, so that cgroup

> > > > subsystems may represent whatever they want with the 16-bit token.
> > > > However, no more than a single subsystem may be able to use this feature
> > > > at the same time.
> > > >
> > > > > I'm tempted to say it might be better to just add a pointer to your
> > > > > throtl_grp structure into struct page_cgroup. Or maybe replace the
> > > > > mem_cgroup pointer with a single pointer to struct css_set. Both of
> > > > > those ideas, though, probably just add unwanted extra overhead now to gain
> > > > > generality which may or may not be wanted in the future.
> > > >
> > > > The pointer to css_set sounds good, but it would add additional space to
> > > > the page_cgroup struct. Now, page_cgroup is 40 bytes (in 64-bit arch)
> > > > and all of them are allocated at boot time. Using unused bits in
> > > > page_cgroup->flags is a choice with no overhead from this point of view.
> > >
> > > I think John suggested replacing mem_cgroup pointer with css_set so that
> > > size of the strcuture does not increase but it leads extra level of
> > > indirection.
> >
> > OK, got it sorry.
> >
> > So, IIUC we save css_set pointer and get a struct cgroup as following:
> >
> >   struct cgroup *cgrp = css_set->subsys[subsys_id]->cgroup;
> >
> > Then, for example to get the mem_cgroup reference:
> >
> >   struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
> >
> > It seems a lot of indirections, but I may have done something wrong or
> > there could be a simpler way to do it.
> >
>
>
> Then, page_cgroup should have reference count on css_set and make tons of
> atomic ops.
>
> BTW, bits of pc->flags are used for storing sectionID or nodeID.
> Please clarify your 16bit never breaks that information. And please keep
> more 4-5 flags for dirty_ratio support of memcg.

OK, I didn't see the recent work about section and node id encoded in
the pc->flags, thanks. So, it'd be probably better to rebase the patch to
the latest mmotm to check all this stuff.

>
> I wonder I can make pc->mem_cgroup to be pc->memid(16bit), then,
> ==

> static inline struct mem_cgroup *get_memcg_from_pc(struct page_cgroup *pc)
> {
> struct cgroup_subsys_state *css = css_lookup(&mem_cgroup_subsys, pc->memid);
> return container_of(css, struct mem_cgroup, css);
> }
> ==
> Overhead will be seen at updating file statistics and LRU management.
>
> But, hmm, can't you do that tracking without page_cgroup ?
> Because the number of dirty/writeback pages are far smaller than total pages,
> chasing I/O with dynamic structure is not very bad..
>
> prepareing [pfn -> blkio] record table and move that information to struct bio
> in dynamic way is very difficult ?

This would be ok for dirty pages, but consider that we're also tracking
anonymous pages. So, if we want to control the swap IO we actually need
to save this information for a lot of pages and at the end I think we'll
basically duplicate the page_cgroup code.

Thanks,
-Andrea

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Vivek Goyal on Wed, 23 Feb 2011 15:23:54 GMT
View Forum Message <> Reply to Message

> > Agreed. Granularity of per inode level might be accetable in many
> > cases. Again, I am worried faster group getting stuck behind slower
> > group.
> >
> > I am wondering if we are trying to solve the problem of ASYNC write throttling
> > at wrong layer. Should ASYNC IO be throttled before we allow task to write to
> > page cache. The way we throttle the process based on dirty ratio, can we
> > just check for throttle limits also there or something like that.(I think
> > that's what you had done in your initial throttling controller implementation?)
>
> Right. This is exactly the same approach I've used in my old throttling
> controller: throttle sync READs and WRITEs at the block layer and async
> WRITEs when the task is dirtying memory pages.
>
> This is probably the simplest way to resolve the problem of faster group
> getting blocked by slower group, but the controller will be a little bit

> more leaky, because the writeback IO will be never throttled and we'll
> see some limited IO spikes during the writeback.

Yes writeback will not be throttled. Not sure how big a problem that is.

- We have controlled the input rate. So that should help a bit.
- May be one can put some high limit on root cgroup to in blkio throttle controller to limit overall WRITE rate of the system.
- For SATA disks, try to use CFQ which can try to minimize the impact of WRITE.

It will atleast provide consistent bandwindth experience to application.

>However, this is always
> a better solution IMHO respect to the current implementation that is
> affected by that kind of priority inversion problem.
>
> I can try to add this logic to the current blk-throttle controller if
> you think it is worth to test it.

At this point of time I have few concerns with this approach.

- Configuration issues. Asking user to plan for SYNC ans ASYNC IO separately is inconvenient. One has to know the nature of workload.

- Most likely we will come up with global limits (atleast to begin with), and not per device limit. That can lead to contention on one single lock and scalability issues on big systems.

Having said that, this approach should reduce the kernel complexity a lot. So if we can do some intelligent locking to limit the overhead then it will boil down to reduced complexity in kernel vs ease of use to user. I guess at this point of time I am inclined towards keeping it simple in kernel.

Couple of people have asked me that we have backup jobs running at night and we want to reduce the IO bandwidth of these jobs to limit the impact on latency of other jobs, I guess this approach will definitely solve that issue.

IMHO, it might be worth trying this approach and see how well does it work. It might not solve all the problems but can be helpful in many situations.

I feel that for proportional bandwidth division, implementing ASYNC control at CFQ will make sense because even if things get serialized in higher layers, consequences are not very bad as it is work conserving algorithm. But for throttling serialization will lead to bad consequences.

May be one can think of new files in blkio controller to limit async IO
per group during page dirty time.

blkio.throttle.async.write_bps_limit
blkio.throttle.async.write_iops_limit

Thanks
Vivek

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Andrea Righi on Wed, 23 Feb 2011 23:14:11 GMT
View Forum Message <> Reply to Message

On Wed, Feb 23, 2011 at 10:23:54AM -0500, Vivek Goyal wrote:
> > > Agreed. Granularity of per inode level might be accetable in many
> > > cases. Again, I am worried faster group getting stuck behind slower
> > > group.
> > >
> > > I am wondering if we are trying to solve the problem of ASYNC write throttling
> > > at wrong layer. Should ASYNC IO be throttled before we allow task to write to
> > > page cache. The way we throttle the process based on dirty ratio, can we
> > > just check for throttle limits also there or something like that.(I think
> > > that's what you had done in your initial throttling controller implementation?)
> >
> > Right. This is exactly the same approach I've used in my old throttling
> > controller: throttle sync READs and WRITEs at the block layer and async
> > WRITEs when the task is dirtying memory pages.
> >
> > This is probably the simplest way to resolve the problem of faster group
> > getting blocked by slower group, but the controller will be a little bit
> > more leaky, because the writeback IO will be never throttled and we'll
> > see some limited IO spikes during the writeback.
>
> Yes writeback will not be throttled. Not sure how big a problem that is.
>
> - We have controlled the input rate. So that should help a bit.
> - May be one can put some high limit on root cgroup to in blkio throttle
>   controller to limit overall WRITE rate of the system.
> - For SATA disks, try to use CFQ which can try to minimize the impact of
>   WRITE.
>
> It will atleast provide consistent bandwindth experience to application.

Right.

>
> >However, this is always
> > a better solution IMHO respect to the current implementation that is
> > affected by that kind of priority inversion problem.
> >
> > I can try to add this logic to the current blk-throttle controller if
> > you think it is worth to test it.
>
> At this point of time I have few concerns with this approach.
>
> - Configuration issues. Asking user to plan for SYNC ans ASYNC IO
> separately is inconvenient. One has to know the nature of workload.
>
> - Most likely we will come up with global limits (atleast to begin with),
> and not per device limit. That can lead to contention on one single
> lock and scalability issues on big systems.
>
> Having said that, this approach should reduce the kernel complexity a lot.
> So if we can do some intelligent locking to limit the overhead then it
> will boil down to reduced complexity in kernel vs ease of use to user. I
> guess at this point of time I am inclined towards keeping it simple in
> kernel.
>

BTW, with this approach probably we can even get rid of the page
tracking stuff for now. If we don't consider the swap IO, any other IO
operation from our point of view will happen directly from process
context (writes in memory + sync reads from the block device).

However, I'm sure we'll need the page tracking also for the blkio
controller soon or later. This is an important information and also the
proportional bandwidth controller can take advantage of it.

>
> Couple of people have asked me that we have backup jobs running at night
> and we want to reduce the IO bandwidth of these jobs to limit the impact
> on latency of other jobs, I guess this approach will definitely solve
> that issue.
>
> IMHO, it might be worth trying this approach and see how well does it work. It
> might not solve all the problems but can be helpful in many situations.

Agreed. This could be a good tradeoff for a lot of common cases.

>
> I feel that for proportional bandwidth division, implementing ASYNC

> control at CFQ will make sense because even if things get serialized in
> higher layers, consequences are not very bad as it is work conserving
> algorithm. But for throttling serialization will lead to bad consequences.

Agreed.

>
> May be one can think of new files in blkio controller to limit async IO
> per group during page dirty time.
>
> blkio.throttle.async.write_bps_limit
> blkio.throttle.async.write_iops_limit

OK, I'll try to add the async throttling logic and use this interface.

-Andrea

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

---

Subject: Re: [PATCH 3/5] page_cgroup: make page tracking available for blkio
Posted by KAMEZAWA Hiroyuki on Wed, 23 Feb 2011 23:58:05 GMT
View Forum Message <> Reply to Message

On Wed, 23 Feb 2011 09:59:11 +0100
Andrea Righi <arighi@develer.com> wrote:

> >
> > I wonder I can make pc->mem_cgroup to be pc->memid(16bit), then,
> > ==
> > static inline struct mem_cgroup *get_memcg_from_pc(struct page_cgroup *pc)
> > {
> > struct cgroup_subsys_state *css = css_lookup(&mem_cgroup_subsys, pc->memid);
> > return container_of(css, struct mem_cgroup, css);
> > }
> > ==
> > Overhead will be seen at updating file statistics and LRU management.
> >
> > But, hmm, can't you do that tracking without page_cgroup ?
> > Because the number of dirty/writeback pages are far smaller than total pages,
> > chasing I/O with dynamic structure is not very bad..
> >
> > prepareing [pfn -> blkio] record table and move that information to struct bio
> > in dynamic way is very difficult ?
>
> This would be ok for dirty pages, but consider that we're also tracking

> anonymous pages. So, if we want to control the swap IO we actually need
> to save this information for a lot of pages and at the end I think we'll
> basically duplicate the page_cgroup code.
>

swap io is always started with bio and the task/mm_struct.
So, if we can record information in bio, no page tracking is required.
You can record information to bio just by reading mm->owner.

Thanks,
-Kame

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Vivek Goyal on Thu, 24 Feb 2011 00:10:33 GMT
View Forum Message <> Reply to Message

On Thu, Feb 24, 2011 at 12:14:11AM +0100, Andrea Righi wrote:
> On Wed, Feb 23, 2011 at 10:23:54AM -0500, Vivek Goyal wrote:
> > > > Agreed. Granularity of per inode level might be accetable in many
> > > > cases. Again, I am worried faster group getting stuck behind slower
> > > > group.
> > > >
> > > > I am wondering if we are trying to solve the problem of ASYNC write throttling
> > > > at wrong layer. Should ASYNC IO be throttled before we allow task to write to
> > > > page cache. The way we throttle the process based on dirty ratio, can we
> > > > just check for throttle limits also there or something like that.(I think
> > > > that's what you had done in your initial throttling controller implementation?)
> > >
> > > Right. This is exactly the same approach I've used in my old throttling
> > > controller: throttle sync READs and WRITEs at the block layer and async
> > > WRITEs when the task is dirtying memory pages.
> > >
> > > This is probably the simplest way to resolve the problem of faster group
> > > getting blocked by slower group, but the controller will be a little bit
> > > more leaky, because the writeback IO will be never throttled and we'll
> > > see some limited IO spikes during the writeback.
> >
> > Yes writeback will not be throttled. Not sure how big a problem that is.
> >
> > - We have controlled the input rate. So that should help a bit.
> > - May be one can put some high limit on root cgroup to in blkio throttle
> > controller to limit overall WRITE rate of the system.

> > - For SATA disks, try to use CFQ which can try to minimize the impact of
> > WRITE.
> >
> > It will atleast provide consistent bandwidth experience to application.
>
> Right.
>
> >
> > >However, this is always
> > > a better solution IMHO respect to the current implementation that is
> > > affected by that kind of priority inversion problem.
> > >
> > > I can try to add this logic to the current blk-throttle controller if
> > > you think it is worth to test it.
> > >
> > At this point of time I have few concerns with this approach.
> >
> > - Configuration issues. Asking user to plan for SYNC ans ASYNC IO
> > separately is inconvenient. One has to know the nature of workload.
> >
> > - Most likely we will come up with global limits (atleast to begin with),
> > and not per device limit. That can lead to contention on one single
> > lock and scalability issues on big systems.
> >
> > Having said that, this approach should reduce the kernel complexity a lot.
> > So if we can do some intelligent locking to limit the overhead then it
> > will boil down to reduced complexity in kernel vs ease of use to user. I
> > guess at this point of time I am inclined towards keeping it simple in
> > kernel.
> >
>
> BTW, with this approach probably we can even get rid of the page
> tracking stuff for now.

Agreed.

> If we don't consider the swap IO, any other IO
> operation from our point of view will happen directly from process
> context (writes in memory + sync reads from the block device).

Why do we need to account for swap IO? Application never asked for swap IO. It is kernel's decision to move soem pages to swap to free up some memory. What's the point in charging those pages to application group and throttle accordingly?

>
> However, I'm sure we'll need the page tracking also for the blkio
> controller soon or later. This is an important information and also the

> proportional bandwidth controller can take advantage of it.

Yes page tracking will be needed for CFQ proportional bandwidth ASYNC
write support. But until and unless we implement memory cgroup dirty
ratio and figure a way out to make writeback logic cgroup aware, till
then I think page tracking stuff is not really useful.

> >
> > Couple of people have asked me that we have backup jobs running at night
> > and we want to reduce the IO bandwidth of these jobs to limit the impact
> > on latency of other jobs, I guess this approach will definitely solve
> > that issue.
> >
> > IMHO, it might be worth trying this approach and see how well does it work. It
> > might not solve all the problems but can be helpful in many situations.
>
> Agreed. This could be a good tradeoff for a lot of common cases.
>
> >
> > I feel that for proportional bandwidth division, implementing ASYNC
> > control at CFQ will make sense because even if things get serialized in
> > higher layers, consequences are not very bad as it is work conserving
> > algorithm. But for throttling serialization will lead to bad consequences.
>
> Agreed.
>
> >
> > May be one can think of new files in blkio controller to limit async IO
> > per group during page dirty time.
> >
> > blkio.throttle.async.write_bps_limit
> > blkio.throttle.async.write_iops_limit
>
> OK, I'll try to add the async throttling logic and use this interface.

Cool, I would like to play with it a bit once patches are ready.

Thanks
Vivek

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by KAMEZAWA Hiroyuki on Thu, 24 Feb 2011 00:40:39 GMT

On Wed, 23 Feb 2011 19:10:33 -0500
Vivek Goyal <vgoyal@redhat.com> wrote:

> On Thu, Feb 24, 2011 at 12:14:11AM +0100, Andrea Righi wrote:
> > On Wed, Feb 23, 2011 at 10:23:54AM -0500, Vivek Goyal wrote:
> > > > > > Agreed. Granularity of per inode level might be accetable in many
> > > > > > cases. Again, I am worried faster group getting stuck behind slower
> > > > > > group.
> > > > > >
> > > > > > I am wondering if we are trying to solve the problem of ASYNC write throttling
> > > > > > at wrong layer. Should ASYNC IO be throttled before we allow task to write to
> > > > > > page cache. The way we throttle the process based on dirty ratio, can we
> > > > > > just check for throttle limits also there or something like that.(I think
> > > > > > that's what you had done in your initial throttling controller implementation?)
> > > > >
> > > > > Right. This is exactly the same approach I've used in my old throttling
> > > > > controller: throttle sync READs and WRITEs at the block layer and async
> > > > > WRITEs when the task is dirtying memory pages.
> > > > >
> > > > > This is probably the simplest way to resolve the problem of faster group
> > > > > getting blocked by slower group, but the controller will be a little bit
> > > > > more leaky, because the writeback IO will be never throttled and we'll
> > > > > see some limited IO spikes during the writeback.
> > > >
> > > > Yes writeback will not be throttled. Not sure how big a problem that is.
> > > >
> > > > - We have controlled the input rate. So that should help a bit.
> > > > - May be one can put some high limit on root cgroup to in blkio throttle
> > > >   controller to limit overall WRITE rate of the system.
> > > > - For SATA disks, try to use CFQ which can try to minimize the impact of
> > > >   WRITE.
> > > >
> > > > It will atleast provide consistent bandwindth experience to application.
> > >
> > > Right.
> > >
> > > >However, this is always
> > > > > a better solution IMHO respect to the current implementation that is
> > > > > affected by that kind of priority inversion problem.
> > > > >
> > > > > I can try to add this logic to the current blk-throttle controller if
> > > > > you think it is worth to test it.
> > > >
> > > > At this point of time I have few concerns with this approach.
> > > >
> > > > - Configuration issues. Asking user to plan for SYNC ans ASYNC IO

> > > separately is inconvenient. One has to know the nature of workload.
> > >
> > > - Most likely we will come up with global limits (atleast to begin with),
> > > and not per device limit. That can lead to contention on one single
> > > lock and scalability issues on big systems.
> > >
> > > Having said that, this approach should reduce the kernel complexity a lot.
> > > So if we can do some intelligent locking to limit the overhead then it
> > > will boil down to reduced complexity in kernel vs ease of use to user. I
> > > guess at this point of time I am inclined towards keeping it simple in
> > > kernel.
> > >
> >
> > BTW, with this approach probably we can even get rid of the page
> > tracking stuff for now.
>
> Agreed.
>
> > If we don't consider the swap IO, any other IO
> > operation from our point of view will happen directly from process
> > context (writes in memory + sync reads from the block device).
>
> Why do we need to account for swap IO? Application never asked for swap
> IO. It is kernel's decision to move soem pages to swap to free up some
> memory. What's the point in charging those pages to application group
> and throttle accordingly?
>

I think swap I/O should be controlled by memcg's dirty_ratio.
But, IIRC, NEC guy had a requirement for this...

I think some enterprise cusotmer may want to throttle the whole speed of
swapout I/O (not swapin)...so, they may be glad if they can limit throttle
the I/O against a disk partition or all I/O tagged as 'swapio' rather than
some cgroup name.

But I'm afraid slow swapout may consume much dirty_ratio and make things
worse ;)

> >
> > However, I'm sure we'll need the page tracking also for the blkio
> > controller soon or later. This is an important information and also the
> > proportional bandwidth controller can take advantage of it.
>
> Yes page tracking will be needed for CFQ proportional bandwidth ASYNC
> write support. But until and unless we implement memory cgroup dirty

> ratio and figure a way out to make writeback logic cgroup aware, till
> then I think page tracking stuff is not really useful.
>

I think Greg Thelen is now preparing patches for dirty_ratio.

Thanks,
-Kame

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Greg Thelen on Thu, 24 Feb 2011 02:01:22 GMT
View Forum Message <> Reply to Message

On Wed, Feb 23, 2011 at 4:40 PM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> On Wed, 23 Feb 2011 19:10:33 -0500
> Vivek Goyal <vgoyal@redhat.com> wrote:
>
>> On Thu, Feb 24, 2011 at 12:14:11AM +0100, Andrea Righi wrote:
>> > On Wed, Feb 23, 2011 at 10:23:54AM -0500, Vivek Goyal wrote:
>> > > > > > > Agreed. Granularity of per inode level might be accetable in many
>> > > > > > > cases. Again, I am worried faster group getting stuck behind slower
>> > > > > > > group.
>> > > > > > >
>> > > > > > > I am wondering if we are trying to solve the problem of ASYNC write throttling
>> > > > > > > at wrong layer. Should ASYNC IO be throttled before we allow task to write to
>> > > > > > > page cache. The way we throttle the process based on dirty ratio, can we
>> > > > > > > just check for throttle limits also there or something like that.(I think
>> > > > > > > that's what you had done in your initial throttling controller implementation?)
>> > > > > >
>> > > > > > Right. This is exactly the same approach I've used in my old throttling
>> > > > > > controller: throttle sync READs and WRITEs at the block layer and async
>> > > > > > WRITEs when the task is dirtying memory pages.
>> > > > > >
>> > > > > > This is probably the simplest way to resolve the problem of faster group
>> > > > > > getting blocked by slower group, but the controller will be a little bit
>> > > > > > more leaky, because the writeback IO will be never throttled and we'll
>> > > > > > see some limited IO spikes during the writeback.
>> > > > >
>> > > > > Yes writeback will not be throttled. Not sure how big a problem that is.
>> > > > >
>> > > > - We have controlled the input rate. So that should help a bit.

>> > > - May be one can put some high limit on root cgroup to in blkio throttle
>> > > controller to limit overall WRITE rate of the system.
>> > > - For SATA disks, try to use CFQ which can try to minimize the impact of
>> > > WRITE.
>> > >
>> > > It will atleast provide consistent bandwindth experience to application.
>> >
>> > Right.
>> >
>> > >
>> > > >However, this is always
>> > > > a better solution IMHO respect to the current implementation that is
>> > > > affected by that kind of priority inversion problem.
>> > > >
>> > > > I can try to add this logic to the current blk-throttle controller if
>> > > > you think it is worth to test it.
>> > > >
>> > > At this point of time I have few concerns with this approach.
>> > > >
>> > > - Configuration issues. Asking user to plan for SYNC ans ASYNC IO
>> > > separately is inconvenient. One has to know the nature of workload.
>> > > >
>> > > - Most likely we will come up with global limits (atleast to begin with),
>> > > and not per device limit. That can lead to contention on one single
>> > > lock and scalability issues on big systems.
>> > > >
>> > > Having said that, this approach should reduce the kernel complexity a lot.
>> > > So if we can do some intelligent locking to limit the overhead then it
>> > > will boil down to reduced complexity in kernel vs ease of use to user. I
>> > > guess at this point of time I am inclined towards keeping it simple in
>> > > kernel.
>> > > >
>> >
>> > BTW, with this approach probably we can even get rid of the page
>> > tracking stuff for now.
>>
>> Agreed.
>>
>> > If we don't consider the swap IO, any other IO
>> > operation from our point of view will happen directly from process
>> > context (writes in memory + sync reads from the block device).
>>
>> Why do we need to account for swap IO? Application never asked for swap
>> IO. It is kernel's decision to move soem pages to swap to free up some
>> memory. What's the point in charging those pages to application group
>> and throttle accordingly?
>>
>

> I think swap I/O should be controlled by memcg's dirty_ratio.
> But, IIRC, NEC guy had a requirement for this...
>
> I think some enterprise cusotmer may want to throttle the whole speed of
> swapout I/O (not swapin)...so, they may be glad if they can limit throttle
> the I/O against a disk partition or all I/O tagged as 'swapio' rather than
> some cgroup name.
>
> But I'm afraid slow swapout may consume much dirty_ratio and make things
> worse ;)
>
>
>
>> >
>> > However, I'm sure we'll need the page tracking also for the blkio
>> > controller soon or later. This is an important information and also the
>> > proportional bandwidth controller can take advantage of it.
>>
>> Yes page tracking will be needed for CFQ proportional bandwidth ASYNC
>> write support. But until and unless we implement memory cgroup dirty
>> ratio and figure a way out to make writeback logic cgroup aware, till
>> then I think page tracking stuff is not really useful.
>>
>
> I think Greg Thelen is now preparing patches for dirty_ratio.
>
> Thanks,
> -Kame
>
>

Correct. I am working on the memcg dirty_ratio patches with latest mmotm memcg. I am running some test cases which should be complete tomorrow. Once testing is complete, I will sent the patches for review.

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

---

Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Balbir Singh on Thu, 24 Feb 2011 06:08:53 GMT
View Forum Message <> Reply to Message

* Andrea Righi <arighi@develer.com> [2011-02-22 18:12:51]:

> Currently the blkio.throttle controller only support synchronous IO requests.

> This means that we always look at the current task to identify the "owner" of
> each IO request.
>
> However dirty pages in the page cache can be wrote to disk 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 perspective.
>
> The same concept is also valid for anonymous pages involed in IO operations
> (swap).
>
> This patch allow to track the cgroup that originally dirtied each page in page
> cache and each anonymous page and pass these informations to the blk-throttle
> controller. These informations can be used to provide a better service level
> differentiation of buffered writes swap IO between different cgroups.
>
> Testcase
> ========
> - create a cgroup with 1MiB/s write limit:
> # mount -t cgroup -o blkio none /mnt/cgroup
> # mkdir /mnt/cgroup/foo
> # echo 8:0 $((1024 * 1024)) > /mnt/cgroup/foo/blkio.throttle.write_bps_device
>
> - move a task into the cgroup and run a dd to generate some writeback IO
>
> Results:
> - 2.6.38-rc6 vanilla:
> $ cat /proc/$$/cgroup
> 1:blkio:/foo
> $ dd if=/dev/zero of=zero bs=1M count=1024 &
> $ dstat -df
> --dsk/sda--
> read writ
> 0 19M
> 0 19M
> 0 0
> 0 0
> 0 19M
> ...
>
> - 2.6.38-rc6 + blk-throttle writeback IO control:
> $ cat /proc/$$/cgroup
> 1:blkio:/foo

> $ dd if=/dev/zero of=zero bs=1M count=1024 &
> $ dstat -df
> --dsk/sda--
> read writ
> 0 1024
> 0 1024
> 0 1024
> 0 1024
> 0 1024
> ...
>

Thanks for looking into this, further review follows.

--
Three Cheers,
Balbir

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