Subject: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups based o
Posted by Satoshi UCHIDA on Thu, 03 Apr 2008 07:09:12 GMT
View Forum Message <> Reply to Message

This patchset modified a name of subsystem (from "cfq_cgroup" to "cfq")
and a checking in create function.

This patchset introduce "Yet Another" I/O bandwidth controlling
subsystem for cgroups based on CFQ (called 2 layer CFQ).

The idea of 2 layer CFQ is to build fairness control per group on the top of existing CFQ control.
We add a new data structure called CFQ meta-data on the top of
cfqd in order to control I/O bandwidth for cgroups.
CFQ meta-data control cfq_datas by service tree (rb-tree) and
CFQ algorithm when synchronous I/O.
An active cfqd controls queue for cfq by service tree.
Namely, the CFQ meta-data control traditional CFQ data.
the CFQ data runs conventionally.

```
          cfqmd     cfqmd     (cfqmd = cfq meta-data)
           |         |
  cfqc  -- cfqd ----- cfqd     (cfqd = cfq data,
           |         |        cfqc = cfq cgroup data)
  cfqc  --[cfqd]----- cfqd
            $B", (B
 $B!!!!!!!!!! (Bconventional control.
```

This patchset is gainst 2.6.25-rc2-mm1.

Last week, we found a patchset from Vasily Tarasov (Open VZ) that
posted to LKML.
  [RFC][PATCH 0/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
  http://lwn.net/Articles/274652/

Our subsystem and Vasily's one are similar on the point of modifying
the CFQ subsystem, but they are different on the point of the layer of
implementation. Vasily's subsystem add a new layer for cgroup between
cfqd and cfqq, but our subsystem add a new layer for cgroup on the top
of cfqd.

The different of implementation from OpenVZ's one are:
  * top layer algorithm is also based on service tree, and
  * top layer program is stored in the different file (block/cfq-cgroup.c).

We hope to discuss not which is better implementation, but what is the
best way to implement I/O bandwidth control based on CFQ here.

Please give us your comments, questions and suggestions.

Finally, we introduce a usage of our implementation.

* Preparation for using 2 layer CFQ

1. Adopt this patchset to kernel 2.6.25-rc2-mm1.

2. Build kernel with CFQ-CGROUP option.

3. Restart new kernel.

4. Mount cfq_cgroup special device to device directory.
   ex.
   mkdir /dev/cgroup
   mount -t cgroup -o cfq cfq /dev/cgroup

* Usage of grouping control.
- Create New group
   Make new directory under /dev/cgroup.
   For example, the following command genrerates a 'test1' group.
      mkdir /dev/cgroup/test1

- Insert task to group
   Write process id(pid) on "tasks" entry in the corresponding group.
   For example, the following command sets task with pid 1100 into test1 group.
      echo 1100 > /dev/cgroup/test1/tasks
   Child tasks of this tasks is also inserted into test1 group.

- Change I/O priority of group
   Write priority on "cfq.ioprio" entry in corresponding group.
   For example, the following command sets priority of rank 2 to 'test1' group.
      echo 2 > /dev/cgroup/test1/tasks
   I/O priority for cgroups takes the value from 0 to 7. It is same as
   existing per-task CFQ.

- Change I/O priority of task
   Use existing "ionice" command.

* Example
Two I/O load (dd command) runs some conditions.

- When they are same group and same priority,

program
```
#!/bin/sh
echo $$ > /dev/cgroup/tasks
echo $$ > /dev/cgroup/test/tasks
ionice -c 2 -n 3 dd if=/internal/data1 of=/dev/null bs=1M count=1K &
ionice -c 2 -n 3 dd if=/internal/data2 of=/dev/null bs=1M count=1K &
echo $$ > /dev/cgroup/test2/tasks
echo $$ > /dev/cgroup/tasks
```

result
```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 27.7676 s, 38.7 MB/s
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 28.8482 s, 37.2 MB/s
```

These tasks was fair, therefore they finished at similar time.

- When they are same group and different priorities (0 and 7),

program
```
#!/bin/sh
echo $$ > /dev/cgroup/tasks
echo $$ > /dev/cgroup/test/tasks
ionice -c 2 -n 0 dd if=/internal/data1 of=/dev/null bs=1M count=1K &
ionice -c 2 -n 7 dd if=/internal/data2 of=/dev/null bs=1M count=1K &
echo $$ > /dev/cgroup/test2/tasks
echo $$ > /dev/cgroup/tasks
```

result
```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 18.8373 s, 57.0 MB/s
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 28.108 s, 38.2 MB/s
```

The first task (copy data1) had high priority, therefore it finished at fast.

- When they are different groups and different priorities (0 and 7),

program
```
#!/bin/sh
echo $$ > /dev/cgroup/tasks
echo $$ > /dev/cgroup/test/tasks
ionice -c 2 -n 0 dd if=/internal/data1 of=/dev/null bs=1M count=1K
echo $$ > /dev/cgroup/test2/tasks
ionice -c 2 -n 7 dd if=/internal/data2 of=/dev/null bs=1M count=1K
echo $$ > /dev/cgroup/tasks
```

result
```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 28.1661 s, 38.1 MB/s
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 28.8486 s, 37.2 MB/s
```

The first task (copy data1) had high priority, but they finished at similar time.
Because their groups had same priority.

- When they are different groups with different priorities (7 and 0)
and same priority,

program
```
#!/bin/sh
echo $$ > /dev/cgroup/tasks
echo 7 > /dev/cgroup/test/cfq.ioprio
echo $$ > /dev/cgroup/test/tasks
ionice -c 2 -n 0 dd if=/internal/data1 of=/dev/null bs=1M count=1K >& test1.log &
echo 0 > /dev/cgroup/test2/cfq.ioprio
echo $$ > /dev/cgroup/test2/tasks
ionice -c 2 -n 7 dd if=/internal/data2 of=/dev/null bs=1M count=1K >& test2.log &
echo $$ > /dev/cgroup/tasks
```

result
```
=== test1.log ===
  1024+0 records in
  1024+0 records out
  1073741824 bytes (1.1 GB) copied, 27.3971 s, 39.2 MB/s
=== test2.log ===
  1024+0 records in
  1024+0 records out
  1073741824 bytes (1.1 GB) copied, 17.3837 s, 61.8 MB/s
```

This first task (copy data1) had high priority, but they finished at late.
Because its group had low priority.

=====
Satoshi UHICDA
NEC Corporation.

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

---

Subject: [RFC][patch 5/12][CFQ-cgroup] Create cfq driver unique data
Posted by Satoshi UCHIDA on Thu, 03 Apr 2008 07:14:06 GMT
View Forum Message <> Reply to Message

This patch exacts driver unique data into new structure(cfq_driver_data)
in order to move top control layer(cfq_meata_data layer
in next patch).

CFQ_DRV_UNIQ_DATA macro calculates control data in top control layer.
In one lalyer CFQ, macro selects cfq_driver_data in cfq_data.
In two lalyer CFQ, macro selects cfq_driver_data in cfq_meta_data.
(in [7/12] patch)

Signed-off-by: Satoshi UCHIDA <uchida@ap.jp.nec.com>

---
block/cfq-iosched.c | 138 ++++++++++++++++++++++++++----------
include/linux/cfq-iosched.h | 48 ++++++++++---
2 files changed, 102 insertions(+), 84 deletions(-)

diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index c1f9da9..aaf5d7e 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -177,7 +177,7 @@ static inline int cfq_bio_sync(struct bio *bio)
static inline void cfq_schedule_dispatch(struct cfq_data *cfqd)
{
if (cfqd->busy_queues)
- kblockd_schedule_work(&cfqd->unplug_work);
+ kblockd_schedule_work(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work);
}

static int cfq_queue_empty(struct request_queue *q)
@@ -260,7 +260,7 @@ cfq_choose_req(struct cfq_data *cfqd, struct request *rq1, struct request
*rq2)
s1 = rq1->sector;

```
  s2 = rq2->sector;

- last = cfqd->last_position;
+ last = CFQ_DRV_UNIQ_DATA(cfqd).last_position;

  /*
   * by definition, 1KiB is 2 sectors
@@ -535,7 +535,7 @@ static void cfq_add_rq_rb(struct request *rq)
   * if that happens, put the alias on the dispatch list
   */
  while ((__alias = elv_rb_add(&cfqq->sort_list, rq)) != NULL)
-  cfq_dispatch_insert(cfqd->queue, __alias);
+  cfq_dispatch_insert(CFQ_DRV_UNIQ_DATA(cfqd).queue, __alias);

  if (!cfq_cfqq_on_rr(cfqq))
   cfq_add_cfqq_rr(cfqd, cfqq);
@@ -579,7 +579,7 @@ static void cfq_activate_request(struct request_queue *q, struct request
*rq)
 {
  struct cfq_data *cfqd = q->elevator->elevator_data;

- cfqd->rq_in_driver++;
+ CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver++;

  /*
   * If the depth is larger 1, it really could be queueing. But lets
@@ -587,18 +587,18 @@ static void cfq_activate_request(struct request_queue *q, struct
request *rq)
   * low queueing, and a low queueing number could also just indicate
   * a SCSI mid layer like behaviour where limit+1 is often seen.
   */
- if (!cfqd->hw_tag && cfqd->rq_in_driver > 4)
-  cfqd->hw_tag = 1;
+ if (!CFQ_DRV_UNIQ_DATA(cfqd).hw_tag && CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver > 4)
+  CFQ_DRV_UNIQ_DATA(cfqd).hw_tag = 1;

- cfqd->last_position = rq->hard_sector + rq->hard_nr_sectors;
+ CFQ_DRV_UNIQ_DATA(cfqd).last_position = rq->hard_sector + rq->hard_nr_sectors;
 }

 static void cfq_deactivate_request(struct request_queue *q, struct request *rq)
 {
  struct cfq_data *cfqd = q->elevator->elevator_data;

- WARN_ON(!cfqd->rq_in_driver);
- cfqd->rq_in_driver--;
+ WARN_ON(!CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver);
+ CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver--;
```

```
 }

 static void cfq_remove_request(struct request *rq)
@@ -706,7 +706,7 @@ __cfq_slice_expired(struct cfq_data *cfqd, struct cfq_queue *cfqq,
       int timed_out)
 {
  if (cfq_cfqq_wait_request(cfqq))
-  del_timer(&cfqd->idle_slice_timer);
+  del_timer(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer);

  cfq_clear_cfqq_must_dispatch(cfqq);
  cfq_clear_cfqq_wait_request(cfqq);
@@ -722,9 +722,9 @@ __cfq_slice_expired(struct cfq_data *cfqd, struct cfq_queue *cfqq,
  if (cfqq == cfqd->active_queue)
   cfqd->active_queue = NULL;

- if (cfqd->active_cic) {
-  put_io_context(cfqd->active_cic->ioc);
-  cfqd->active_cic = NULL;
+ if (CFQ_DRV_UNIQ_DATA(cfqd).active_cic) {
+  put_io_context(CFQ_DRV_UNIQ_DATA(cfqd).active_cic->ioc);
+  CFQ_DRV_UNIQ_DATA(cfqd).active_cic = NULL;
  }
 }

@@ -763,15 +763,15 @@ static struct cfq_queue *cfq_set_active_queue(struct cfq_data *cfqd)
 static inline sector_t cfq_dist_from_last(struct cfq_data *cfqd,
       struct request *rq)
 {
- if (rq->sector >= cfqd->last_position)
-  return rq->sector - cfqd->last_position;
+ if (rq->sector >= CFQ_DRV_UNIQ_DATA(cfqd).last_position)
+  return rq->sector - CFQ_DRV_UNIQ_DATA(cfqd).last_position;
  else
-  return cfqd->last_position - rq->sector;
+  return CFQ_DRV_UNIQ_DATA(cfqd).last_position - rq->sector;
 }

 static inline int cfq_rq_close(struct cfq_data *cfqd, struct request *rq)
 {
- struct cfq_io_context *cic = cfqd->active_cic;
+ struct cfq_io_context *cic = CFQ_DRV_UNIQ_DATA(cfqd).active_cic;

  if (!sample_valid(cic->seek_samples))
   return 0;
@@ -804,13 +804,13 @@ static void cfq_arm_slice_timer(struct cfq_data *cfqd)
  /*
   * idle is disabled, either manually or by past process history
```

```
   */
- if (!cfqd->cfq_slice_idle || !cfq_cfqq_idle_window(cfqq))
+ if (!CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle || !cfq_cfqq_idle_window(cfqq))
   return;

  /*
   * task has exited, don't wait
   */
- cic = cfqd->active_cic;
+ cic = CFQ_DRV_UNIQ_DATA(cfqd).active_cic;
  if (!cic || !atomic_read(&cic->ioc->nr_tasks))
   return;

@@ -829,11 +829,11 @@ static void cfq_arm_slice_timer(struct cfq_data *cfqd)
   * fair distribution of slice time for a process doing back-to-back
   * seeks. so allow a little bit of time for him to submit a new rq
   */
- sl = cfqd->cfq_slice_idle;
+ sl = CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle;
  if (sample_valid(cic->seek_samples) && CIC_SEEKY(cic))
   sl = min(sl, msecs_to_jiffies(CFQ_MIN_TT));

- mod_timer(&cfqd->idle_slice_timer, jiffies + sl);
+ mod_timer(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer, jiffies + sl);
 }

 /*
@@ -849,7 +849,7 @@ static void cfq_dispatch_insert(struct request_queue *q, struct request
*rq)
  elv_dispatch_sort(q, rq);

  if (cfq_cfqq_sync(cfqq))
-  cfqd->sync_flight++;
+  CFQ_DRV_UNIQ_DATA(cfqd).sync_flight++;
 }

 /*
@@ -918,7 +918,7 @@ static struct cfq_queue *cfq_select_queue(struct cfq_data *cfqd)
   * flight or is idling for a new request, allow either of these
   * conditions to happen (or time out) before selecting a new queue.
   */
- if (timer_pending(&cfqd->idle_slice_timer) ||
+ if (timer_pending(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer) ||
      (cfqq->dispatched && cfq_cfqq_idle_window(cfqq))) {
   cfqq = NULL;
   goto keep_queue;
@@ -957,13 +957,13 @@ __cfq_dispatch_requests(struct cfq_data *cfqd, struct cfq_queue
*cfqq,
```

```
   /*
    * finally, insert request into driver dispatch list
    */
-  cfq_dispatch_insert(cfqd->queue, rq);
+  cfq_dispatch_insert(CFQ_DRV_UNIQ_DATA(cfqd).queue, rq);

   dispatched++;

-  if (!cfqd->active_cic) {
+  if (!CFQ_DRV_UNIQ_DATA(cfqd).active_cic) {
    atomic_inc(&RQ_CIC(rq)->ioc->refcount);
-   cfqd->active_cic = RQ_CIC(rq);
+   CFQ_DRV_UNIQ_DATA(cfqd).active_cic = RQ_CIC(rq);
   }

   if (RB_EMPTY_ROOT(&cfqq->sort_list))
@@ -990,7 +990,7 @@ static int __cfq_forced_dispatch_cfqq(struct cfq_queue *cfqq)
  int dispatched = 0;

  while (cfqq->next_rq) {
-  cfq_dispatch_insert(cfqq->cfqd->queue, cfqq->next_rq);
+  cfq_dispatch_insert(CFQ_DRV_UNIQ_DATA(cfqq->cfqd).queue, cfqq->next_rq);
   dispatched++;
  }

@@ -1044,12 +1044,12 @@ static int cfq_dispatch_requests(struct request_queue *q, int force)
     break;
   }

-  if (cfqd->sync_flight && !cfq_cfqq_sync(cfqq))
+  if (CFQ_DRV_UNIQ_DATA(cfqd).sync_flight && !cfq_cfqq_sync(cfqq))
    break;

   cfq_clear_cfqq_must_dispatch(cfqq);
   cfq_clear_cfqq_wait_request(cfqq);
-  del_timer(&cfqd->idle_slice_timer);
+  del_timer(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer);

   dispatched += __cfq_dispatch_requests(cfqd, cfqq, max_dispatch);
  }
@@ -1175,7 +1175,7 @@ static void cfq_exit_single_io_context(struct io_context *ioc,
  struct cfq_data *cfqd = cic->key;

  if (cfqd) {
-  struct request_queue *q = cfqd->queue;
+  struct request_queue *q = CFQ_DRV_UNIQ_DATA(cfqd).queue;
   unsigned long flags;
```

```
   spin_lock_irqsave(q->queue_lock, flags);
@@ -1200,7 +1200,7 @@ cfq_alloc_io_context(struct cfq_data *cfqd, gfp_t gfp_mask)
  struct cfq_io_context *cic;

  cic = kmem_cache_alloc_node(cfq_ioc_pool, gfp_mask | __GFP_ZERO,
-       cfqd->queue->node);
+     CFQ_DRV_UNIQ_DATA(cfqd).queue->node);
  if (cic) {
   cic->last_end_request = jiffies;
   INIT_LIST_HEAD(&cic->queue_list);
@@ -1265,7 +1265,7 @@ static void changed_ioprio(struct io_context *ioc, struct cfq_io_context
*cic)
  if (unlikely(!cfqd))
   return;

- spin_lock_irqsave(cfqd->queue->queue_lock, flags);
+ spin_lock_irqsave(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);

  cfqq = cic->cfqq[ASYNC];
  if (cfqq) {
@@ -1281,7 +1281,7 @@ static void changed_ioprio(struct io_context *ioc, struct cfq_io_context
*cic)
  if (cfqq)
   cfq_mark_cfqq_prio_changed(cfqq);

- spin_unlock_irqrestore(cfqd->queue->queue_lock, flags);
+ spin_unlock_irqrestore(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
 }

 static void cfq_ioc_set_ioprio(struct io_context *ioc)
@@ -1313,16 +1313,16 @@ retry:
     * the allocator to do whatever it needs to attempt to
     * free memory.
     */
-   spin_unlock_irq(cfqd->queue->queue_lock);
+   spin_unlock_irq(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock);
    new_cfqq = kmem_cache_alloc_node(cfq_pool,
      gfp_mask | __GFP_NOFAIL | __GFP_ZERO,
-     cfqd->queue->node);
-   spin_lock_irq(cfqd->queue->queue_lock);
+     CFQ_DRV_UNIQ_DATA(cfqd).queue->node);
+   spin_lock_irq(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock);
    goto retry;
   } else {
    cfqq = kmem_cache_alloc_node(cfq_pool,
      gfp_mask | __GFP_ZERO,
-     cfqd->queue->node);
+     CFQ_DRV_UNIQ_DATA(cfqd).queue->node);
```

```
    if (!cfqq)
     goto out;
   }
@@ -1494,9 +1494,9 @@ static int cfq_cic_link(struct cfq_data *cfqd, struct io_context *ioc,
   radix_tree_preload_end();

   if (!ret) {
-   spin_lock_irqsave(cfqd->queue->queue_lock, flags);
+   spin_lock_irqsave(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
    list_add(&cic->queue_list, &cfqd->cic_list);
-   spin_unlock_irqrestore(cfqd->queue->queue_lock, flags);
+   spin_unlock_irqrestore(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
   }
  }

@@ -1519,7 +1519,7 @@ cfq_get_io_context(struct cfq_data *cfqd, gfp_t gfp_mask)

  might_sleep_if(gfp_mask & __GFP_WAIT);

- ioc = get_io_context(gfp_mask, cfqd->queue->node);
+ ioc = get_io_context(gfp_mask, CFQ_DRV_UNIQ_DATA(cfqd).queue->node);
  if (!ioc)
   return NULL;

@@ -1551,7 +1551,7 @@ static void
 cfq_update_io_thinktime(struct cfq_data *cfqd, struct cfq_io_context *cic)
 {
  unsigned long elapsed = jiffies - cic->last_end_request;
- unsigned long ttime = min(elapsed, 2UL * cfqd->cfq_slice_idle);
+ unsigned long ttime = min(elapsed, 2UL * CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle);

  cic->ttime_samples = (7*cic->ttime_samples + 256) / 8;
  cic->ttime_total = (7*cic->ttime_total + 256*ttime) / 8;
@@ -1604,11 +1604,11 @@ cfq_update_idle_window(struct cfq_data *cfqd, struct cfq_queue
*cfqq,

  enable_idle = cfq_cfqq_idle_window(cfqq);

- if (!atomic_read(&cic->ioc->nr_tasks) || !cfqd->cfq_slice_idle ||
-     (cfqd->hw_tag && CIC_SEEKY(cic)))
+ if (!atomic_read(&cic->ioc->nr_tasks) || !CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle ||
+     (CFQ_DRV_UNIQ_DATA(cfqd).hw_tag && CIC_SEEKY(cic)))
   enable_idle = 0;
  else if (sample_valid(cic->ttime_samples)) {
-  if (cic->ttime_mean > cfqd->cfq_slice_idle)
+  if (cic->ttime_mean > CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle)
    enable_idle = 0;
   else
```

```
    enable_idle = 1;
@@ -1657,7 +1657,7 @@ cfq_should_preempt(struct cfq_data *cfqd, struct cfq_queue
*new_cfqq,
  if (rq_is_meta(rq) && !cfqq->meta_pending)
   return 1;

- if (!cfqd->active_cic || !cfq_cfqq_wait_request(cfqq))
+ if (!CFQ_DRV_UNIQ_DATA(cfqd).active_cic || !cfq_cfqq_wait_request(cfqq))
   return 0;

  /*
@@ -1717,8 +1717,8 @@ cfq_rq_enqueued(struct cfq_data *cfqd, struct cfq_queue *cfqq,
    */
   if (cfq_cfqq_wait_request(cfqq)) {
    cfq_mark_cfqq_must_dispatch(cfqq);
-   del_timer(&cfqd->idle_slice_timer);
-   blk_start_queueing(cfqd->queue);
+   del_timer(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer);
+   blk_start_queueing(CFQ_DRV_UNIQ_DATA(cfqd).queue);
   }
  } else if (cfq_should_preempt(cfqd, cfqq, rq)) {
   /*
@@ -1728,7 +1728,7 @@ cfq_rq_enqueued(struct cfq_data *cfqd, struct cfq_queue *cfqq,
    */
   cfq_preempt_queue(cfqd, cfqq);
   cfq_mark_cfqq_must_dispatch(cfqq);
-  blk_start_queueing(cfqd->queue);
+  blk_start_queueing(CFQ_DRV_UNIQ_DATA(cfqd).queue);
  }
 }

@@ -1755,16 +1755,16 @@ static void cfq_completed_request(struct request_queue *q, struct
request *rq)

  now = jiffies;

- WARN_ON(!cfqd->rq_in_driver);
+ WARN_ON(!CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver);
  WARN_ON(!cfqq->dispatched);
- cfqd->rq_in_driver--;
+ CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver--;
  cfqq->dispatched--;

  if (cfq_cfqq_sync(cfqq))
-  cfqd->sync_flight--;
+  CFQ_DRV_UNIQ_DATA(cfqd).sync_flight--;

  if (!cfq_class_idle(cfqq))
```

```
-  cfqd->last_end_request = now;
+  CFQ_DRV_UNIQ_DATA(cfqd).last_end_request = now;

  if (sync)
   RQ_CIC(rq)->last_end_request = now;
@@ -1784,7 +1784,7 @@ static void cfq_completed_request(struct request_queue *q, struct
request *rq)
    cfq_arm_slice_timer(cfqd);
  }

- if (!cfqd->rq_in_driver)
+ if (!CFQ_DRV_UNIQ_DATA(cfqd).rq_in_driver)
   cfq_schedule_dispatch(cfqd);
 }

@@ -1928,9 +1928,7 @@ queue_fail:

 static void cfq_kick_queue(struct work_struct *work)
 {
- struct cfq_data *cfqd =
-  container_of(work, struct cfq_data, unplug_work);
- struct request_queue *q = cfqd->queue;
+ struct request_queue *q =  __cfq_container_of_queue(work);
  unsigned long flags;

  spin_lock_irqsave(q->queue_lock, flags);
@@ -1948,7 +1946,7 @@ static void cfq_idle_slice_timer(unsigned long data)
  unsigned long flags;
  int timed_out = 1;

- spin_lock_irqsave(cfqd->queue->queue_lock, flags);
+ spin_lock_irqsave(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);

  cfqq = cfqd->active_queue;
  if (cfqq) {
@@ -1980,13 +1978,13 @@ expire:
 out_kick:
  cfq_schedule_dispatch(cfqd);
 out_cont:
- spin_unlock_irqrestore(cfqd->queue->queue_lock, flags);
+ spin_unlock_irqrestore(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
 }

 static void cfq_shutdown_timer_wq(struct cfq_data *cfqd)
 {
- del_timer_sync(&cfqd->idle_slice_timer);
- kblockd_flush_work(&cfqd->unplug_work);
+ del_timer_sync(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer);
```

```
+ kblockd_flush_work(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work);
 }

 static void cfq_put_async_queues(struct cfq_data *cfqd)
@@ -2007,7 +2005,7 @@ static void cfq_put_async_queues(struct cfq_data *cfqd)
 static void cfq_exit_queue(elevator_t *e)
 {
  struct cfq_data *cfqd = e->elevator_data;
- struct request_queue *q = cfqd->queue;
+ struct request_queue *q = CFQ_DRV_UNIQ_DATA(cfqd).queue;

  cfq_shutdown_timer_wq(cfqd);

@@ -2044,15 +2042,15 @@ static void *cfq_init_queue(struct request_queue *q)
  cfqd->service_tree = CFQ_RB_ROOT;
  INIT_LIST_HEAD(&cfqd->cic_list);

- cfqd->queue = q;
+ CFQ_DRV_UNIQ_DATA(cfqd).queue = q;

- init_timer(&cfqd->idle_slice_timer);
- cfqd->idle_slice_timer.function = cfq_idle_slice_timer;
- cfqd->idle_slice_timer.data = (unsigned long) cfqd;
+ init_timer(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer);
+ CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer.function = cfq_idle_slice_timer;
+ CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer.data = (unsigned long) cfqd;

- INIT_WORK(&cfqd->unplug_work, cfq_kick_queue);
+ INIT_WORK(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work, cfq_kick_queue);

- cfqd->last_end_request = jiffies;
+ CFQ_DRV_UNIQ_DATA(cfqd).last_end_request = jiffies;
  cfqd->cfq_quantum = cfq_quantum;
  cfqd->cfq_fifo_expire[0] = cfq_fifo_expire[0];
  cfqd->cfq_fifo_expire[1] = cfq_fifo_expire[1];
@@ -2061,7 +2059,7 @@ static void *cfq_init_queue(struct request_queue *q)
  cfqd->cfq_slice[0] = cfq_slice_async;
  cfqd->cfq_slice[1] = cfq_slice_sync;
  cfqd->cfq_slice_async_rq = cfq_slice_async_rq;
- cfqd->cfq_slice_idle = cfq_slice_idle;
+ CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle = cfq_slice_idle;

  return cfqd;
 }
@@ -2122,7 +2120,7 @@ SHOW_FUNCTION(cfq_fifo_expire_sync_show,
cfqd->cfq_fifo_expire[1], 1);
 SHOW_FUNCTION(cfq_fifo_expire_async_show, cfqd->cfq_fifo_expire[0], 1);
 SHOW_FUNCTION(cfq_back_seek_max_show, cfqd->cfq_back_max, 0);
```

```
 SHOW_FUNCTION(cfq_back_seek_penalty_show, cfqd->cfq_back_penalty, 0);
-SHOW_FUNCTION(cfq_slice_idle_show, cfqd->cfq_slice_idle, 1);
+SHOW_FUNCTION(cfq_slice_idle_show, CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle, 1);
 SHOW_FUNCTION(cfq_slice_sync_show, cfqd->cfq_slice[1], 1);
 SHOW_FUNCTION(cfq_slice_async_show, cfqd->cfq_slice[0], 1);
 SHOW_FUNCTION(cfq_slice_async_rq_show, cfqd->cfq_slice_async_rq, 0);
@@ -2152,7 +2150,7 @@ STORE_FUNCTION(cfq_fifo_expire_async_store,
&cfqd->cfq_fifo_expire[0], 1,
 STORE_FUNCTION(cfq_back_seek_max_store, &cfqd->cfq_back_max, 0, UINT_MAX, 0);
 STORE_FUNCTION(cfq_back_seek_penalty_store, &cfqd->cfq_back_penalty, 1,
   UINT_MAX, 0);
-STORE_FUNCTION(cfq_slice_idle_store, &cfqd->cfq_slice_idle, 0, UINT_MAX, 1);
+STORE_FUNCTION(cfq_slice_idle_store, &CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle, 0,
UINT_MAX, 1);
 STORE_FUNCTION(cfq_slice_sync_store, &cfqd->cfq_slice[1], 1, UINT_MAX, 1);
 STORE_FUNCTION(cfq_slice_async_store, &cfqd->cfq_slice[0], 1, UINT_MAX, 1);
 STORE_FUNCTION(cfq_slice_async_rq_store, &cfqd->cfq_slice_async_rq, 1,
diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index cce3993..035bfc4 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -19,30 +19,43 @@ struct cfq_rb_root {
 };
 #define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }

+
+#define CFQ_DRV_UNIQ_DATA(cfqd) ((cfqd)->cfq_driv_d)
+
 /*
- * Per block device queue structure
+ * Driver unique data
  */
-struct cfq_data {
+struct cfq_driver_data {
  struct request_queue *queue;

- /*
-  * rr list of queues with requests and the count of them
-  */
- struct cfq_rb_root service_tree;
- unsigned int busy_queues;
-
  int rq_in_driver;
  int sync_flight;
  int hw_tag;

+ struct cfq_io_context *active_cic;
+ struct work_struct unplug_work;
```

```
+
+ sector_t last_position;
+ unsigned long last_end_request;
+
  /*
   * idle window management
   */
  struct timer_list idle_slice_timer;
- struct work_struct unplug_work;
+ unsigned int cfq_slice_idle;
+};
+
+/*
+ * Per block device queue structure
+ */
+struct cfq_data {
+ /*
+  * rr list of queues with requests and the count of them
+  */
+ struct cfq_rb_root service_tree;
+ unsigned int busy_queues;

  struct cfq_queue *active_queue;
- struct cfq_io_context *active_cic;

  /*
   * async queue for each priority case
@@ -50,9 +63,6 @@ struct cfq_data {
  struct cfq_queue *async_cfqq[2][IOPRIO_BE_NR];
  struct cfq_queue *async_idle_cfqq;

- sector_t last_position;
- unsigned long last_end_request;
-
  /*
   * tunables, see top of file
   */
@@ -62,9 +72,19 @@ struct cfq_data {
  unsigned int cfq_back_max;
  unsigned int cfq_slice[2];
  unsigned int cfq_slice_async_rq;
- unsigned int cfq_slice_idle;

  struct list_head cic_list;
+
+ struct cfq_driver_data cfq_driv_d;
+};
+
```

```
+static inline struct request_queue * __cfq_container_of_queue(struct work_struct *work) {
+ struct cfq_driver_data *cfqdd =
+  container_of(work, struct cfq_driver_data, unplug_work);
+ struct cfq_data *cfqd =
+  container_of(cfqdd, struct cfq_data, cfq_driv_d);
+
+ return cfqd->cfq_driv_d.queue;
 };

 #endif  /* _LINUX_CFQ_IOSCHED_H */
--
1.5.4.1
```

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas

Posted by Ryo Tsuruta on Fri, 25 Apr 2008 09:54:44 GMT

View Forum Message <> Reply to Message

Hi,

I report benchmark results of the following I/O bandwidth controllers.

> From: Vasily Tarasov <vtaras@openvz.org>
> Subject: [RFC][PATCH 0/9] cgroups: block: cfq: I/O bandwidth
> controlling subsystem for CGroups based on CFQ
> Date: Fri, 15 Feb 2008 01:53:34 -0500
>
> From: "Satoshi UCHIDA" <s-uchida@ap.jp.nec.com>
> Subject: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth
> controlling subsystem for CGroups based on CFQ
> Date: Thu, 3 Apr 2008 16:09:12 +0900

The test procedure is as follows:
- o Prepare 3 partitions sdc2, sdc3 and sdc4.
- o Run 100 processes issuing random direct I/O with 4KB data on each partitions.
- o Run 3 tests:
  - #1 issuing read I/O only.
  - #2 issuing write I/O only.
  - #3 sdc2 and sdc3 are read, sdc4 is write.
- o Count up the number of I/Os which have done in 60 seconds.

Unfortunately, both bandwidth controllers didn't work as I expected,
On the test #3, the write I/O ate up the bandwidth regardless of the
specified priority level.

Vasily's scheduler
The number of I/Os (percentage to total I/Os)

partition	sdc2	sdc3	sdc4	total
priority	7(highest)	4	0(lowest)	I/Os
#1 read	3620(35.6%)	3474(34.2%)	3065(30.2%)	10159
#2 write	21985(36.6%)	19274(32.1%)	18856(31.4%)	60115
#3 read&write	5571(7.5%)	3253(4.4%)	64977(88.0%)	73801

Satoshi's scheduler
The number of I/Os (percentage to total I/O)

partition	sdc2	sdc3	sdc4	total
priority	0(highest)	4	7(lowest)	I/Os
#1 read	4523(47.8%)	3733(39.5%)	1204(12.7%)	9460
#2 write	65202(59.0%)	35603(32.2%)	9673(8.8%)	110478
#3 read&write	5328(23.0%)	4153(17.9%)	13694(59.1%)	23175

I'd like to see other benchmark results if anyone has.

Thanks,
Ryo Tsuruta

Subject: Re: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroup
Posted by Florian Westphal on Fri, 25 Apr 2008 21:37:02 GMT
View Forum Message <> Reply to Message

Ryo Tsuruta <ryov@valinux.co.jp> wrote:
[..]
> I'd like to see other benchmark results if anyone has.

Here are a few results. IO is issued in 4k chunks,
using O_DIRECT. Each process issues both reads
and writes. There are 60 such processes in each cgroup (except
where noted). Numbers given show the total count of io requests
(read and write) completed in 60 seconds. All processes use
the same partition, fs is ext3.

Vasily's scheduler:

cgroup	s0	s1	total
priority	4	4	I/Os
	24953	24062	49015
	29558(60 processes)	14639 (30 proc)	44197
priority	0	4	
	24221	24047	48268
priority	1	4	
	24897	24509	49406
priority	2	4	
	23295	23622	46917
priority	0	7	
	22301	23373	45674

Satoshi's scheduler:

cgroup	s0	s1	total
priority	3	3	I/Os
	25175	26463	51638
	26944 (60)	26698 (30)	53642
priority	0	3	
	60821	19846	80667
priority	1	3	
	50608	25994	76602
priority	2	3	
	32132	26641	58773
priority	7	0	
	91387	12547	103934

So in short, i can't see any effect when i use Vasily's
i/o scheduler. Setting
echo 10 > /sys/block/hda/queue/iosched/cgrp_slice
did at least show different results in the 'prio 7 vs. prio 0 case'
(~29000 (prio 7) vs. 20000 (prio 0)).

What i found surprising is that Satoshis scheduler has
about twice of the io count...

Thanks, Florian

Subject: Re: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroup
Posted by Ryo Tsuruta on Tue, 29 Apr 2008 00:44:38 GMT
View Forum Message <> Reply to Message

Hi Florian,

> Here are a few results.

Thank you very much.

> What i found surprising is that Satoshis scheduler has
> about twice of the io count...

Of my previous results, Satoshi's scheduler has about twice of the io count on the write test, but both io counts on the read test are nearly the same.

Here are another results. The test procedure is as follows:
- o Prepare 3 partitions sdc2, sdc3 and sdc4.
- o Run 50 processes for sdc2, 100 processes for sdc3 and 150 processes for sdc4. Apply I/O loads in inverse to the priority level.
- o Each process issuing random read/write direct I/O with 4KB data.
- o Count up the number of I/Os which have done in 60 seconds.

The number of I/Os (percentage to total I/Os)

partition	sdc2	sdc3	sdc4	total
processes	50	100	150	I/Os
Normal	2281(18%)	4287(34%)	6005(48%)	12573
Vasily's sched. cgroup priority	7(highest)	4	0(lowest)	
	3713(24%)	6587(42%)	5354(34%)	15654
Satoshi's sched. cgroup priority	0(highest)	4	7(lowest)	
	5399(42%)	6506(50%)	1034(8%)	12939

Satoshi's scheduler suppressed the I/O to the lowest priority partition better than Vasily's one, but Vasily's scheduler got the highest total I/Os.

Thanks,
Ryo Tsuruta

Subject: RE: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Satoshi UCHIDA on Fri, 09 May 2008 10:17:44 GMT
View Forum Message <> Reply to Message

Hi, Ryo-San.
Thank you for your test results.

In the test #2 and #3, did you use direct write?
I guess you have used the non-direct write I/O (using cache).

CFQ I/O scheduler was extended in my and Vasily's controllers so that both controllers inherit the features of CFQ.

The current CFQ I/O scheduler cannot control non-direct write I/Os.
This main cause is a cache system.
Bio data is created by special daemon process, such as pdflush or kswapd, for the write I/O using cache.
Therefore, many non-direct write I/Os will belong to one of cgroup (perhaps, root of cgroup).

We consider that this problem should be resolved by fixing cache system.
Specifically, I/Os created by collection of cache pages belong to I/O-context for task which wrote data to cache.
This resolution has a problem.
 * Who is the owner of cache page?
    Cache is reused by many tasks.
    Therefore, it is difficult to decide owner.

In the test #3, It seems that system could control I/Os only among read (sdc2 and sdc3).
Therefore, your test shows that our controller can control I/O without above problem.

Meanwhile, I'm very interested in the result of your test #2.
In the non-direct write I/O, performance will be influenced to task scheduling and sequence of output pages.
Therefore, non-direct write I/O will be fair in the current default task scheduler.
However, your result shows almost fair in Vasilly's controller, whereas non fair in ours.
I'm just wondering if this is an accidental result or an usual result.

Thanks,
  Satoshi UCHIDA.

> -----Original Message-----
> From: Ryo Tsuruta [mailto:ryov@valinux.co.jp]
> Sent: Friday, April 25, 2008 6:55 PM
> To: s-uchida@ap.jp.nec.com; vtaras@openvz.org
> Cc: linux-kernel@vger.kernel.org;

> containers@lists.linux-foundation.org; axboe@kernel.dk;
> tom-sugawara@ap.jp.nec.com; m-takahashi@ex.jp.nec.com; devel@openvz.org
> Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth
> controlling subsystem for CGroups based on CFQ
>
> Hi,
>
> I report benchmark results of the following I/O bandwidth controllers.
>
> From: Vasily Tarasov <vtaras@openvz.org>
> Subject: [RFC][PATCH 0/9] cgroups: block: cfq: I/O bandwidth
> controlling subsystem for CGroups based on CFQ
> Date: Fri, 15 Feb 2008 01:53:34 -0500
>
> From: "Satoshi UCHIDA" <s-uchida@ap.jp.nec.com>
> Subject: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth
> controlling subsystem for CGroups based on CFQ
> Date: Thu, 3 Apr 2008 16:09:12 +0900
>
> The test procedure is as follows:
> o Prepare 3 partitions sdc2, sdc3 and sdc4.
> o Run 100 processes issuing random direct I/O with 4KB data on each
> partitions.
> o Run 3 tests:
> #1 issuing read I/O only.
> #2 issuing write I/O only.
> #3 sdc2 and sdc3 are read, sdc4 is write.
> o Count up the number of I/Os which have done in 60 seconds.
>
> Unfortunately, both bandwidth controllers didn't work as I expected,
> On the test #3, the write I/O ate up the bandwidth regardless of the
> specified priority level.
>
> Vasily's scheduler
> The number of I/Os (percentage to total I/Os)
>
> ---
> | partition | sdc2 | sdc3 | sdc4 | total
> |
> | priority | 7(highest) | 4 | 0(lowest) | I/Os
> |
>
> |---+---+---+---|---
> |
> | #1 read | 3620(35.6%) | 3474(34.2%) | 3065(30.2%) | 10159
> |
> | #2 write | 21985(36.6%) | 19274(32.1%) | 18856(31.4%) | 60115
> |

> | #3 read&write | 5571(7.5%) | 3253(4.4%) | 64977(88.0%) | 73801
> |
>
> --
>
>   Satoshi's scheduler
>   The number of I/Os (percentage to total I/O)
>
> --
> | partition | sdc2 | sdc3 | sdc4 | total
> |
> | priority | 0(highest) | 4 | 7(lowest) | I/Os
> |
>
> |---------------+--------------+--------------+--------------|--------
> |
> | #1 read | 4523(47.8%) | 3733(39.5%) | 1204(12.7%) | 9460
> |
> | #2 write | 65202(59.0%) | 35603(32.2%) | 9673(8.8%) | 110478
> |
> | #3 read&write | 5328(23.0%) | 4153(17.9%) | 13694(59.1%) | 23175
> |
>
> --
>
> I'd like to see other benchmark results if anyone has.
>
> Thanks,
> Ryo Tsuruta

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Ryo Tsuruta on Mon, 12 May 2008 03:10:19 GMT
View Forum Message <> Reply to Message

Hi, Uchida-san,
Thanks for your comments.

> In the test #2 and #3, did you use direct write?
> I guess you have used the non-direct write I/O (using cache).

Yes, I did. I used direct write in all tests.
I would appreciate it if you would try a test like I did.

--
Ryo Tsuruta <ryov@valinux.co.jp>

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Ryo Tsuruta on Mon, 12 May 2008 15:33:16 GMT
View Forum Message <> Reply to Message

Hi, Uchida-san,

> > In the test #2 and #3, did you use direct write?
> > I guess you have used the non-direct write I/O (using cache).
>
> Yes, I did. I used direct write in all tests.
> I would appreciate it if you would try a test like I did.

And I'll retest and report you back.

--
Ryo Tsuruta <ryov@valinux.co.jp>

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

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Ryo Tsuruta on Thu, 22 May 2008 13:04:38 GMT
View Forum Message <> Reply to Message

Hi Uchida-san,

I realized that the benchmark results which I posted on Apr 25 had
some problems with the testing environment.

  From: Ryo Tsuruta <ryov@valinux.co.jp>
  Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O
  bandwidth controlling subsystem for CGroups based on CFQ
  Date: Fri, 25 Apr 2008 18:54:44 +0900 (JST)

Uchida-san said,

> In the test #2 and #3, did you use direct write?
> I guess you have used the non-direct write I/O (using cache).

I answered "Yes," but actually I did not use direct write I/O, because
I ran these tests on Xen-HVM. Xen-HVM backend driver doesn't use direct
I/O for actual disk operations even though guest OS uses direct I/O.

An another problem was that the CPU time was used up during the tests.

So, I retested with the new testing environment and got good results.
The number of I/Os is proportioned according to the priority levels.

Details of the tests are as follows:

Envirionment:
```
  Linux version 2.6.25-rc2-mm1 based.
  CPU0: Intel(R) Core(TM)2 CPU         6600  @ 2.40GHz stepping 06
  CPU1: Intel(R) Core(TM)2 CPU         6600  @ 2.40GHz stepping 06
  Memory: 2063568k/2088576k available (2085k kernel code, 23684k
  reserved, 911k data, 240k init, 1171072k highmem)
  scsi 1:0:0:0: Direct-Access     ATA      WDC WD2500JS-55N 10.0 PQ: 0  ANSI: 5
  sd 1:0:0:0: [sdb] 488397168 512-byte hardware sectors (250059 MB)
  sd 1:0:0:0: [sdb] Write Protect is off
  sd 1:0:0:0: [sdb] Mode Sense: 00 3a 00 00
  sd 1:0:0:0: [sdb] Write cache: disabled, read cache: enabled,
  doesn't support DPO or FUA
  sdb: sdb1 sdb2 sdb3 sdb4 < sdb5 sdb6 sdb7 sdb8 sdb9 sdb10 sdb11
  sdb12 sdb13 sdb14 sdb15 >
```

Procedures:
- o Prepare 3 partitions sdb5, sdb6 and sdb7.
- o Run 100 processes issuing random direct I/O with 4KB data on each partitions.
- o Run 3 tests:
  - #1 issuing read I/O only.
  - #2 issuing write I/O only.
  - #3 sdb5 and sdb6 are read, sdb7 is write.
- o Count up the number of I/Os which have done in 60 seconds.

Results:

Vasily's scheduler

The number of I/Os (percentage to total I/Os)

partition	sdb5	sdb6	sdb7	total
priority	7(highest)	4	0(lowest)	I/Os
#1 read	3383(35%)	3164(33%)	3142(32%)	9689
#2 write	3017(42%)	2372(33%)	1851(26%)	7240
#3 read&write	4300(36%)	3127(27%)	1521(17%)	8948

Satoshi's scheduler

The number of I/Os (percentage to total I/O)

partition	sdb5	sdb6	sdb7	total
priority	0(highest)	4	7(lowest)	I/Os

```
|---------------+--------------+--------------+--------------|--------|
| #1 read       |   3907(47%)  |   3126(38%)  |   1260(15%)  |  8293  |
| #2 write      |   3389(41%)  |   3024(36%)  |   1901(23%)  |  8314  |
| #3 read&write |   5028(53%)  |   3961(42%)  |    441( 5%)  |  9430  |
 ---------------------------------------------------------------------
```

Thanks,
Ryo Tsuruta

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

---

Subject: RE: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Satoshi UCHIDA on Fri, 23 May 2008 02:53:50 GMT
View Forum Message <> Reply to Message

Hi, Tsuruta-san,

Thanks for your test.

>
> Uchida-san said,
>
> > In the test #2 and #3, did you use direct write?
> > I guess you have used the non-direct write I/O (using cache).
>
> I answered "Yes," but actually I did not use direct write I/O, because
> I ran these tests on Xen-HVM. Xen-HVM backend driver doesn't use direct
> I/O for actual disk operations even though guest OS uses direct I/O.
>

Where did you build expanded CFQ schedulers?
I guess that schedulers can be control I/Os if it is built on guest OS,
But not if on Dom0.
I guess you built on Dom0 so that you could not control I/O. (maybe, you say)

> So, I retested with the new testing environment and got good results.
> The number of I/Os is proportioned according to the priority levels.
>

Ok.
I'm testing both systems and get similar results.
I will report my test in next week.

> Details of the tests are as follows:
>
> Envirionment:
> Linux version 2.6.25-rc2-mm1 based.
> CPU0: Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz stepping 06
> CPU1: Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz stepping 06
> Memory: 2063568k/2088576k available (2085k kernel code, 23684k
> reserved, 911k data, 240k init, 1171072k highmem)
> scsi 1:0:0:0: Direct-Access ATA WDC WD2500JS-55N 10.0 PQ: 0
> ANSI: 5
> sd 1:0:0:0: [sdb] 488397168 512-byte hardware sectors (250059 MB)
> sd 1:0:0:0: [sdb] Write Protect is off
> sd 1:0:0:0: [sdb] Mode Sense: 00 3a 00 00
> sd 1:0:0:0: [sdb] Write cache: disabled, read cache: enabled,
> doesn't support DPO or FUA
> sdb: sdb1 sdb2 sdb3 sdb4 < sdb5 sdb6 sdb7 sdb8 sdb9 sdb10 sdb11
> sdb12 sdb13 sdb14 sdb15 >
>
> Procedures:
> o Prepare 3 partitions sdb5, sdb6 and sdb7.
> o Run 100 processes issuing random direct I/O with 4KB data on each
> partitions.
> o Run 3 tests:
> #1 issuing read I/O only.
> #2 issuing write I/O only.
> #3 sdb5 and sdb6 are read, sdb7 is write.
> o Count up the number of I/Os which have done in 60 seconds.
>
> Results:
> Vasily's scheduler
> The number of I/Os (percentage to total I/Os)
>
> ---
> | partition | sdb5 | sdb6 | sdb7 | total
> |
> | priority | 7(highest) | 4 | 0(lowest) | I/Os
> |
>
> |--------------+-------------+-------------+-------------|--------
> |
> | #1 read | 3383(35%) | 3164(33%) | 3142(32%) | 9689
> |
> | #2 write | 3017(42%) | 2372(33%) | 1851(26%) | 7240
> |
> | #3 read&write | 4300(36%) | 3127(27%) | 1521(17%) | 8948
> |

>
> ----------
>
>   Satoshi's scheduler
>   The number of I/Os (percentage to total I/O)
>
> ----------
> | partition | sdb5 | sdb6 | sdb7 | total
> |
> | priority | 0(highest) | 4 | 7(lowest) | I/Os
> |
>
> |--------------+------------+------------+------------|--------
> |
> | #1 read | 3907(47%) | 3126(38%) | 1260(15%) | 8293
> |
> | #2 write | 3389(41%) | 3024(36%) | 1901(23%) | 8314
> |
> | #3 read&write | 5028(53%) | 3961(42%) | 441(5%) | 9430
> |
>
> ----------
>
> Thanks,
> Ryo Tsuruta

Thanks,
Satoshi UCHIDA.

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Ryo Tsuruta on Mon, 26 May 2008 02:46:27 GMT
View Forum Message <> Reply to Message

Hi Uchida-san,

Thanks for your comment.

> Where did you build expanded CFQ schedulers?
> I guess that schedulers can be control I/Os if it is built on guest OS,
> But not if on Dom0.

> I guess you built on Dom0 so that you could not control I/O. (maybe, you say)

I built them on guest OS.

> I'm testing both systems and get similar results.
> I will report my test in next week.

I'm looking forward to your report.

Thanks,
Ryo Tsuruta

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

---

Subject: RE: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Satoshi UCHIDA on Tue, 27 May 2008 11:32:48 GMT
View Forum Message <> Reply to Message

Hi, Tsuruta-san.

> I'm looking forward to your report.
>

I report my tests.

My test shows following features.

 o The guaranteeing degrees are widely in write I/Os than in read I/Os for each environment.
 o Vasily's scheduler can guarantee I/O control.
    However, its guaranteeing degree is narrow.
      (in particular, at low priority)
 o Satoshi's scheduler can guarantee I/O control.
   However, guaranteeing degree is too small in write only and low priority case.
 o Write I/Os are faster than read I/Os.
   And, CFQ scheduler controls I/Os by time slice.
   So, guaranteeing degree is caused difference from estimating degree at requests level by the situation of read and write I/Os.

I'll continue testing variously.
I hope to improve I/O control scheduler through many tests.

Details of the tests are as follows:

Environment:
  Linux version 2.6.25-rc5-mm1 based.
    4 type:
      kernel with Vasily's scheduler
      kernel with Satoshi's scheduler
      Native kernel
      Native kernel and use ionice commands to each process.

  CPU0: Intel(R) Core(TM)2 CPU 6700 @ 2.66GHz stepping 6
  CPU1: Intel(R) Core(TM)2 CPU 6700 @ 2.66GHz stepping 6
  Memory: 4060180k/5242880k available (2653k kernel code, 132264k reserved, 1412k data, 356k init)
  scsi 3:0:0:0: Direct-Access ATA WDC WD2500JS-19N 10.0 PQ: 0 ANSI: 5
  sd 3:0:0:0: [sdb] 488282256 512-byte hardware sectors (250001 MB)
  sd 3:0:0:0: [sdb] Write Protect is off
sd 3:0:0:0: [sdb] Mode Sense: 00 3a 00 00
  sd 3:0:0:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
  sd 3:0:0:0: [sdb] 488282256 512-byte hardware sectors (250001 MB)
  sd 3:0:0:0: [sdb] Write Protect is off
  sd 3:0:0:0: [sdb] Mode Sense: 00 3a 00 00
  sd 3:0:0:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
  sdb: sdb1 sdb2 sdb3 sdb4 < sdb5 >

Test 1:
 Procedures:
  o Prepare 200 files which size is 250MB on 1 partition sdb3
  o Create 3 groups with priority 0, 4 and 7.
      Estimate peformance in Vasily's scheduler : group-1 57.1% group-2 35.7% group-3 7.2%
      Estimate peformance in Satoshi's scheduler : group-1 61.5% group-2 30.8% group-3 7.7%
      Estimate peformance in Native Linux CFQ scheduler with ionice command :
                                  group-1 92.8% group-2 5.8% group-3 1.4%
  o Run many processes issuing random direct I/O with 4KB data on each files in 3 groups.
      #1 Run 25 processes issuing read I/O only per groups.
      #2 Run 25 processes issuing write I/O only per groups.
      #3 Run 15 processes issuing read I/O and 15 processes issuing writeI/O only per groups.
  o Count up the number of I/Os which have done in 120 seconds.
  o Measure at 5 times in each situation.
  o Results is calculated by averatve of 5 times.

Results:
                  Vasily's scheduler
          The number of I/Os (percentage to total I/Os)
  ----
group	group 1	group 2	group 3	total
priority	7(highest)	4	0(lowest)	I/Os
--------------+----------------+----------------+----------------	---------

#1 Read	6077.2 (46.33%)	3680.8 (28.06%)	3360.2 (25.61%)	13118.2
#2 Write	10291.2 (53.13%)	5282.8 (27.27%)	3796.2 (19.60%)	19370.2
#3 Read&Write	7218.0 (49.86%)	4273.0 (29.52%)	2986.0 (20.63%)	14477.0
	(Read 45.52%)	(Read 51.96%)	(Read 52.63%)	(Read 48.89%)

Satoshi's scheduler
The number of I/Os (percentage to total I/O)

group priority	group 1 0(highest)	group 2 4	group 3 7(lowest)	total I/Os
#1 Read	9082.2 (60.90%)	4403.0 (29.53%)	1427.4 (9.57%)	14912.6
#2 Write	15449.0 (68.74%)	6144.2 (27.34%)	881.8 (3.92%)	22475.0
#3 Read&Write	11283.6 (65.35%)	4699.0 (27.21%)	1284.8 (7.44%)	17267.4
	(Read 41.08%)	(Read 47.84%)	(Read 57.07%)	(Read 44.11%)

Native Linux CFQ scheduler
The number of I/Os (percentage to total I/O)

group	group 1	group 2	group 3	total
#1 Read	4362.2 (34.94%)	3864.4 (30.95%)	4259.8 (34.12%)	12486.4
#2 Write	6918.4 (37.23%)	5894.0 (31.71%)	5772.0 (31.06%)	18584.4
#3 Read&Write	4701.2 (33.62%)	4788.0 (34.24%)	4496.0 (32.15%)	13985.2
	(Read 45.85%)	(Read 48.99%)	(Read 51.28%)	(Read 48.67%)

Native Linux CFQ scheduler with ionice command
The number of I/Os (percentage to total I/O)

group priority	group 1 0(highest)	group 2 4	group 3 7(lowest)	total I/Os
#1 Read	12844.2 (85.34%)	1544.8 (10.26%)	661.8 (4.40%)	15050.8
#2 Write	24858.4 (92.44%)	1568.4 (5.83%)	463.4 (1.72%)	26890.2
#3 Read&Write	16205.4 (85.53%)	2016.8 (10.64%)	725.6 (3.83%)	18947.8
	(Read 37.49%)	(Read 57.97%)	(Read 56.62%)	(Read 40.40%)

Test 2:
Procedures:
- o Prepare 200 files which size is 250MB on 1 partition sdb3
- o Create 3 groups with priority 0, 4 and 7.
- o Run many processes issuing random direct I/O with 4KB data on each files in 3 groups.

#1 Run 25 processes issuing read I/O only in group 1 and group 2 and run 25 processes issuing write I/O only in group 3.
(This pattern is represent by "R-R-W".)
#2 Run 25 processes issuing read I/O only in group 1 and group 3 and run 25 processes issuing write I/O only in group 2.
(This pattern is represent by "R-W-R".)
#3 Run 25 processes issuing read I/O only in group 2 and group 3 and run 25 processes issuing write I/O only in group 1.
(This pattern is represent by "R-R-W".)

- Count up the number of I/Os which have done in 120 seconds.
- Measure at 5 times in each situation.
- Results is calculated by averatve of 5 times.

Results:

Vasily's scheduler
The number of I/Os (percentage to total I/Os)

group priority	group 1 7(highest)	group 2 4	group 3 0(lowest)	total I/Os
#1 R-R-W	8828.2 (52.46%)	4372.2 (25.98%)	3628.8 (21.56%)	16829.2
#2 R-W-R	5510.2 (35.01%)	6584.6 (41.83%)	3646.0 (23.16%)	15740.8
#3 W-R-R	6400.4 (41.91%)	3856.4 (25.25%)	5016.4 (32.84%)	15273.2

Results shows peculiar case in test #2.
I/O counts in group 2 are 5911, 4895, 8498, 9300 and 4319.
In third and fourth time, I/O counts are huge.
An average of first, second and fifth results is following.

group priority	group 1 7(highest)	group 2 4	group 3 0(lowest)	total I/Os
#2 R-W-R	6285.7 (41.82%)	5041.7 (33.54%)	3702.7 (24.64%)	15030.0

$B!! (B

Satoshi's scheduler
The number of I/Os (percentage to total I/O)

group priority	group 1 0(highest)	group 2 4	group 3 7(lowest)	total I/Os
#1 R-R-W	9178.6 (61.95%)	4510.8 (30.44%)	1127.4 (7.61%)	14816.8
#2 R-W-R	9398.0 (56.29%)	6152.2 (36.85%)	1146.4 (6.87%)	16696.6
#3 W-R-R	15527.0 (72.38%)	4544.8 (21.19%)	1380.0 (6.43%)	21451.8

Native Linux CFQ scheduler
The number of I/Os (percentage to total I/O)

group	group 1	group 2	group 3	total
#1 R-R-W	4622.4 (33.15%)	4739.2 (33.98%)	4583.4 (32.87%)	13945.0
#2 R-W-R	4610.6 (31.72%)	5502.2 (37.85%)	4422.4 (30.43%)	14535.2
#3 W-R-R	5518.0 (37.57%)	4734.2 (32.24%)	4433.2 (30.19%)	14685.4

Native Linux CFQ scheduler with ionice
The number of I/Os (percentage to total I/O)

group	group 1	group 2	group 3	total
priority	0(highest)	4	7(lowest)	I/Os
#1 Read	12619.4 (84.49%)	1537.4 (10.29%)	779.2 (5.22%)	14936.0
#2 Write	12724.4 (81.44%)	2276.6 (14.57%)	623.2 (3.99%)	15624.2
#3 Read&Write	24442.8 (91.75%)	1592.6 (5.98%)	604.6 (2.27%)	26640.0

Thanks,
Satoshi UCHIDA.

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Andrea Righi on Fri, 30 May 2008 10:37:25 GMT
View Forum Message <> Reply to Message

Satoshi UCHIDA wrote:
> Hi, Tsuruta-san.
>
>> I'm looking forward to your report.

Hi Satoshi,

I'm testing your patch agains latest Linus git and I've got the
following bug. It can be easily reproduced creating a cgroup, switching
the i/o scheduler from cfq to any other and switch back to cfq again.

-Andrea

BUG: unable to handle kernel paging request at ffffffeb
IP: [<c0212dc6>] cfq_cgroup_sibling_tree_add+0x36/0x90
Oops: 0000 [#1] SMP
Modules linked in: i2c_piix4 ne2k_pci 8390 i2c_core

Pid: 3543, comm: bash Not tainted (2.6.26-rc4 #1)
EIP: 0060:[<c0212dc6>] EFLAGS: 00010286 CPU: 0
EIP is at cfq_cgroup_sibling_tree_add+0x36/0x90
EAX: 00000003 EBX: c7704c90 ECX: ffffffff EDX: c7102180
ESI: c7102240 EDI: c7704c80 EBP: c7afbe94 ESP: c7afbe80
 DS: 007b ES: 007b FS: 00d8 GS: 0033 SS: 0068
Process bash (pid: 3543, ti=c7afa000 task=c7aeda00 task.ti=c7afa000)
Stack: c7704c90 c71022d0 c7ace078 c7704c80 c71020c0 c7afbea8 c021363c
c7ace078
       c7803460 c71020c0 c7afbebc c021360a c7102184 c71020c0 c7102180
c7afbee4
       c02134e0 00000000 c7102184 c72b8ab0 00000001 c7102140 c72b8ab0
c04b8ac0
Call Trace:
 [<c021363c>] ? cfq_cgroup_init_cfq_data+0x7c/0x80
 [<c021360a>] ? cfq_cgroup_init_cfq_data+0x4a/0x80
 [<c02134e0>] ? __cfq_cgroup_init_queue+0x100/0x1e0
 [<c021097b>] ? cfq_init_queue+0xb/0x10
 [<c0204ff8>] ? elevator_init_queue+0x8/0x10
 [<c0205cd0>] ? elv_iosched_store+0x80/0x2b0
 [<c0209379>] ? queue_attr_store+0x49/0x70
 [<c01c488b>] ? sysfs_write_file+0xbb/0x110
 [<c0186276>] ? vfs_write+0x96/0x160
 [<c01c47d0>] ? sysfs_write_file+0x0/0x110
 [<c018696d>] ? sys_write+0x3d/0x70
 [<c0104267>] ? sysenter_past_esp+0x78/0xd1
 =======================
Code: ec 08 8b 82 90 00 00 00 83 e0 fc 89 45 f0 8d 82 90 00 00 00 39 45
f0 75 5f
8d 47 10 89 45 ec 89 c3 31 c0 eb 11 8b 56 7c 8d 41 04 <3b> 51 ec 8d 59
08 0f 43
d8 89 c8 8b 0b 85 c9 75 e9 89 86 90 00
EIP: [<c0212dc6>] cfq_cgroup_sibling_tree_add+0x36/0x90 SS:ESP
0068:c7afbe80
---[end trace 9701f4859bb53d27]---

_____

Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Ryo Tsuruta on Tue, 03 Jun 2008 08:15:35 GMT
View Forum Message <> Reply to Message

Hi Uchida-san,

> I report my tests.

I did a similar test to yours. I increased the number of I/Os
which are issued simultaneously up to 100 per cgroup.

Procedures:
- o Prepare 300 files which size is 250MB on 1 partition sdb3
- o Create three groups with priority 0, 4 and 7.
- o Run many processes issuing random direct I/O with 4KB data on each files in three groups.
  - #1 Run 25 processes issuing read I/O only per group.
  - #2 Run 100 processes issuing read I/O only per group.
- o Count up the number of I/Os which have done in 10 minutes.

The number of I/Os (percentage to total I/O)

group	group 1	group 2	group 3	total
priority	0(highest)	4	7(lowest)	I/Os
Estimate				
Performance	61.5%	30.8%	7.7%	
#1 25procs	52763(57%)	30811(33%)	9575(10%)	93149
#2 100procs	24949(40%)	21325(34%)	16508(26%)	62782

The result of test #1 is close to your estimation, but the result
of test #2 is not, the gap between the estimation and the result
increased.

In addition, I got the following message during test #2. Program
"ioload", our benchmark program, was blocked more than 120 seconds.
Do you see any problems?

INFO: task ioload:8456 blocked for more than 120 seconds.
"echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.

```
ioload       D 00000008  2772  8456   8419
       f72eb740 00200082 c34862c0 00000008 c3565170 c35653c0 c2009d80
       00000001
       c1d1bea0 00200046 ffffffff f6ee039c 00000000 00000000 00000000
       c2009d80
       018db000 00000000 f71a6a00 c0604fb6 00000000 f71a6bc8 c04876a4
       00000000
Call Trace:
 [<c0604fb6>] io_schedule+0x4a/0x81
 [<c04876a4>] __blockdev_direct_IO+0xa04/0xb54
 [<c04a3aa2>] ext2_direct_IO+0x35/0x3a
 [<c04a4757>] ext2_get_block+0x0/0x603
 [<c044ab81>] generic_file_direct_IO+0x103/0x118
 [<c044abe6>] generic_file_direct_write+0x50/0x13d
 [<c044b59e>] __generic_file_aio_write_nolock+0x375/0x4c3
 [<c046e571>] link_path_walk+0x86/0x8f
 [<c044a1e8>] find_lock_page+0x19/0x6d
 [<c044b73e>] generic_file_aio_write+0x52/0xa9
 [<c0466256>] do_sync_write+0xbf/0x100
 [<c042ca44>] autoremove_wake_function+0x0/0x2d
 [<c0413366>] update_curr+0x83/0x116
 [<c0605280>] mutex_lock+0xb/0x1a
 [<c04b653b>] security_file_permission+0xc/0xd
 [<c0466197>] do_sync_write+0x0/0x100
 [<c046695d>] vfs_write+0x83/0xf6
 [<c0466ea9>] sys_write+0x3c/0x63
 [<c04038de>] syscall_call+0x7/0xb
 [<c0600000>] print_cpu_info+0x27/0x92
 =======================
```

Thanks,
Ryo Tsuruta

Subject: RE: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups bas
Posted by Satoshi UCHIDA on Thu, 26 Jun 2008 04:49:20 GMT
View Forum Message <> Reply to Message

Hi, Tsuruta.

> In addition, I got the following message during test #2. Program
> "ioload", our benchmark program, was blocked more than 120 seconds.
> Do you see any problems?

No.
I tried to test in environment which runs from 1 to 200 processes
per group.
However, such message was not output.

> The result of test #1 is close to your estimation, but the result
> of test #2 is not, the gap between the estimation and the result
> increased.

In the above my test, the gap between the estimation and the result
is increasing as a process increases.

And, in native CFQ with ionice command, this situation is a similar.
These circumstances are shown in the case of more than processes of total 200.

I'll investigate this problem continuously.

Thanks,
  Satoshi Uchida.

> -----Original Message-----
> From: Ryo Tsuruta [mailto:ryov@valinux.co.jp]
> Sent: Tuesday, June 03, 2008 5:16 PM
> To: s-uchida@ap.jp.nec.com
> Cc: axboe@kernel.dk; vtaras@openvz.org;
> containers@lists.linux-foundation.org; tom-sugawara@ap.jp.nec.com;
> linux-kernel@vger.kernel.org
> Subject: Re: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth
> controlling subsystem for CGroups based on CFQ
>
> Hi Uchida-san,
>
> > I report my tests.
>
> I did a similar test to yours. I increased the number of I/Os
> which are issued simultaneously up to 100 per cgroup.
>
>   Procedures:
>   o Prepare 300 files which size is 250MB on 1 partition sdb3
>   o Create three groups with priority 0, 4 and 7.
>   o Run many processes issuing random direct I/O with 4KB data on each
>   files in three groups.
>   #1 Run 25 processes issuing read I/O only per group.
>   #2 Run 100 processes issuing read I/O only per group.
>   o Count up the number of I/Os which have done in 10 minutes.
>

>   The number of I/Os (percentage to total I/O)
>   ---
> | group | group 1 | group 2 | group 3 | total |
> | priority | 0(highest) | 4 | 7(lowest) | I/Os |
> |-------------+------------+------------+------------+---------|
> | Estimate | | | | |
> | Performance | 61.5% | 30.8% | 7.7% | |
> |-------------+------------+------------+------------|---------|
> | #1 25procs | 52763(57%) | 30811(33%) | 9575(10%) | 93149 |
> | #2 100procs | 24949(40%) | 21325(34%) | 16508(26%) | 62782 |
>   ---
>
> The result of test #1 is close to your estimation, but the result
> of test #2 is not, the gap between the estimation and the result
> increased.
>
> In addition, I got the following message during test #2. Program
> "ioload", our benchmark program, was blocked more than 120 seconds.
> Do you see any problems?
>
> INFO: task ioload:8456 blocked for more than 120 seconds.
> "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
> ioload   D 00000008 2772 8456   8419
>   f72eb740 00200082 c34862c0 00000008 c3565170 c35653c0 c2009d80
>   00000001
>   c1d1bea0 00200046 ffffffff f6ee039c 00000000 00000000 00000000
>   c2009d80
>   018db000 00000000 f71a6a00 c0604fb6 00000000 f71a6bc8 c04876a4
>   00000000
> Call Trace:
> [<c0604fb6>] io_schedule+0x4a/0x81
> [<c04876a4>] __blockdev_direct_IO+0xa04/0xb54
> [<c04a3aa2>] ext2_direct_IO+0x35/0x3a
> [<c04a4757>] ext2_get_block+0x0/0x603
> [<c044ab81>] generic_file_direct_IO+0x103/0x118
> [<c044abe6>] generic_file_direct_write+0x50/0x13d
> [<c044b59e>] __generic_file_aio_write_nolock+0x375/0x4c3
> [<c046e571>] link_path_walk+0x86/0x8f
> [<c044a1e8>] find_lock_page+0x19/0x6d
> [<c044b73e>] generic_file_aio_write+0x52/0xa9
> [<c0466256>] do_sync_write+0xbf/0x100
> [<c042ca44>] autoremove_wake_function+0x0/0x2d
> [<c0413366>] update_curr+0x83/0x116
> [<c0605280>] mutex_lock+0xb/0x1a
> [<c04b653b>] security_file_permission+0xc/0xd
> [<c0466197>] do_sync_write+0x0/0x100
> [<c046695d>] vfs_write+0x83/0xf6
> [<c0466ea9>] sys_write+0x3c/0x63

> [<c04038de>] syscall_call+0x7/0xb
> [<c0600000>] print_cpu_info+0x27/0x92
> ======================
>
> Thanks,
> Ryo Tsuruta

_____

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