Subject: [RFC][patch 0/11][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups based on CF
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:22:00 GMT
View Forum Message <> Reply to Message

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_cgroup cfq_cgroup /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_cgroup.io_prio" 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 $$ > /dev/cgroup/tasks
  echo 7 > /dev/cgroup/test/cfq_cgroup.ioprio
  echo $$ > /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_cgroup.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 inmorre
    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.

Regards,
==========
Satoshi UCHIDA <s-uchida@ap.jp.nec.com>

NEC Coropration.

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

Subject: [RFC][patch 1/11][CFQ-cgroup] Add Configuration
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:27:31 GMT
View Forum Message <> Reply to Message

This patch adds configuration entry into block/Kconfig.iosched.

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

```
diff --git a/block/Kconfig.iosched b/block/Kconfig.iosched
index 96a01b3..493c68c 100644
--- a/block/Kconfig.iosched
+++ b/block/Kconfig.iosched
@@ -25,6 +25,15 @@ config IOSCHED_CFQ

    If unsure, say Y.

+config CGROUP_CFQ
+ bool "handling cgroup in CFQ"
+ default n
+ depends on IOSCHED_CFQ && CGROUPS
+ ---help---
+         This option exptends CFQ to handling cgroup.
+         CFQ is changed into two layler control
+         per-cgroup layler and per-task layler.
+
 config IOSCHED_AS
  tristate "Anticipatory I/O scheduler"
  default y
```

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

Subject: [RFC][patch 2/11][CFQ-cgroup] Move header file
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:30:17 GMT
View Forum Message <> Reply to Message

This patch moves some data structure into header file
(include/linux/cfq-iosched.h).

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

```
diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index 0f962ec..c1f9da9 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -11,6 +11,7 @@
 #include <linux/elevator.h>
 #include <linux/rbtree.h>
 #include <linux/ioprio.h>
+#include <linux/cfq-iosched.h>

 /*
  * tunables
@@ -58,65 +59,6 @@ static struct completion *ioc_gone;

 #define sample_valid(samples) ((samples) > 80)

-/*
- * Most of our rbtree usage is for sorting with min extraction, so
- * if we cache the leftmost node we don't have to walk down the tree
- * to find it. Idea borrowed from Ingo Molnars CFS scheduler. We should
- * move this into the elevator for the rq sorting as well.
- */
-struct cfq_rb_root {
- struct rb_root rb;
- struct rb_node *left;
-};
-#define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }
-
-/*
- * Per block device queue structure
- */
-struct cfq_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;
```

```
-
- /*
-  * idle window management
-  */
- struct timer_list idle_slice_timer;
- struct work_struct unplug_work;
-
- struct cfq_queue *active_queue;
- struct cfq_io_context *active_cic;
-
- /*
-  * async queue for each priority case
-  */
- 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
-  */
- unsigned int cfq_quantum;
- unsigned int cfq_fifo_expire[2];
- unsigned int cfq_back_penalty;
- 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;
-};

 /*
  * Per process-grouping structure
diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
new file mode 100644
index 0000000..cce3993
--- /dev/null
+++ b/include/linux/cfq-iosched.h
@@ -0,0 +1,70 @@
+#ifndef _LINUX_CFQ_IOSCHED_H
+#define _LINUX_CFQ_IOSCHED_H
+
+#include <linux/rbtree.h>
+#include <linux/list.h>
+
+struct request_queue;
```

```
+struct cfq_io_context;
+
+/*
+ * Most of our rbtree usage is for sorting with min extraction, so
+ * if we cache the leftmost node we don't have to walk down the tree
+ * to find it. Idea borrowed from Ingo Molnars CFS scheduler. We should
+ * move this into the elevator for the rq sorting as well.
+ */
+struct cfq_rb_root {
+ struct rb_root rb;
+ struct rb_node *left;
+};
+#define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }
+
+/*
+ * Per block device queue structure
+ */
+struct cfq_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;
+
+ /*
+  * idle window management
+  */
+ struct timer_list idle_slice_timer;
+ struct work_struct unplug_work;
+
+ struct cfq_queue *active_queue;
+ struct cfq_io_context *active_cic;
+
+ /*
+  * async queue for each priority case
+  */
+ 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
+  */
+ unsigned int cfq_quantum;
+ unsigned int cfq_fifo_expire[2];
+ unsigned int cfq_back_penalty;
+ 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;
+};
+
+#endif  /* _LINUX_CFQ_IOSCHED_H */
```

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

---

Subject: [RFC][patch 3/11][CFQ-cgroup] Introduce cgroup subsystem
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:32:29 GMT
View Forum Message <> Reply to Message

This patch introduces a simple cgroup subsystem.
New cgroup subsystem is called cfq_cgroup.

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

```
diff --git a/block/Makefile b/block/Makefile
index 5a43c7d..ea07b46 100644
--- a/block/Makefile
+++ b/block/Makefile
@@ -11,6 +11,7 @@ obj-$(CONFIG_IOSCHED_NOOP) += noop-iosched.o
 obj-$(CONFIG_IOSCHED_AS) += as-iosched.o
 obj-$(CONFIG_IOSCHED_DEADLINE) += deadline-iosched.o
 obj-$(CONFIG_IOSCHED_CFQ) += cfq-iosched.o
+obj-$(CONFIG_CGROUP_CFQ) += cfq-cgroup.o

 obj-$(CONFIG_BLK_DEV_IO_TRACE) += blktrace.o
 obj-$(CONFIG_BLOCK_COMPAT) += compat_ioctl.o
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
new file mode 100644
index 0000000..cea2b92
--- /dev/null
+++ b/block/cfq-cgroup.c
```

```
@@ -0,0 +1,57 @@
+/*
+ *  CFQ CGROUP disk scheduler.
+ *
+ *     This program is a wrapper program that is
+ *     extend CFQ disk scheduler for handling
+ *     cgroup subsystem.
+ *
+ *     This program is based on original CFQ code.
+ *
+ *  Copyright (C) 2008 Satoshi UCHIDA <s-uchida@ap.jp.nec.com>
+ *   and NEC Corp.
+ */
+
+#include <linux/blkdev.h>
+#include <linux/cgroup.h>
+#include <linux/cfq-iosched.h>
+
+struct cfq_cgroup {
+ struct cgroup_subsys_state css;
+};
+
+
+static inline struct cfq_cgroup *cgroup_to_cfq_cgroup(struct cgroup *cont)
+{
+ return container_of(cgroup_subsys_state(cont, cfq_cgroup_subsys_id),
+       struct cfq_cgroup, css);
+}
+
+static struct cgroup_subsys_state *
+cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct cfq_cgroup *cfqc;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return ERR_PTR(-EPERM);
+
+ if (!cgroup_is_descendant(cont))
+  return ERR_PTR(-EPERM);
+
+ cfqc = kzalloc(sizeof(struct cfq_cgroup), GFP_KERNEL);
+ if (unlikely(!cfqc))
+  return ERR_PTR(-ENOMEM);
+
+ return &cfqc->css;
+}
+
+static void cfq_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
+{
+ kfree(cgroup_to_cfq_cgroup(cont));
+}
+
+struct cgroup_subsys cfq_cgroup_subsys = {
+ .name = "cfq_cgroup",
+ .create = cfq_cgroup_create,
+ .destroy = cfq_cgroup_destroy,
+ .subsys_id = cfq_cgroup_subsys_id,
+};
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index 1ddebfc..5d2e991 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_CFQ
+SUBSYS(cfq_cgroup)
+#endif
+
+/* */
```

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

Subject: [RFC][patch 4/11][CFQ-cgroup] Create cfq driver unique data
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:33:52 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 [6/11] patch)

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

```
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 */
```

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

Subject: [RFC][patch 5/11][CFQ-cgroup] Add cfq optional operation framework
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:35:05 GMT
View Forum Message <> Reply to Message

This patch creates a cfq optional operations framework.
This framework defines specific functions for expanding CFQ.

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

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index b5303d9..95663f9 100644
--- a/block/cfq-cgroup.c
+++ b/block/cfq-cgroup.c
@@ -151,3 +151,7 @@ struct cgroup_subsys cfq_cgroup_subsys = {
  .subsys_id = cfq_cgroup_subsys_id,
  .populate = cfq_cgroup_populate,
 };
```

```
+
+
+struct cfq_ops opt = {
+};
diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index aaf5d7e..245c252 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -2233,6 +2233,11 @@ static void __exit cfq_exit(void)
 module_init(cfq_init);
 module_exit(cfq_exit);

+#ifndef CONFIG_CGROUP_CFQ
+struct cfq_ops opt = {
+};
+#endif
+
 MODULE_AUTHOR("Jens Axboe");
 MODULE_LICENSE("GPL");
 MODULE_DESCRIPTION("Completely Fair Queueing IO scheduler");
diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index 035bfc4..9287da1 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -87,4 +87,10 @@ static inline struct request_queue * __cfq_container_of_queue(struct
work_struct
  return cfqd->cfq_driv_d.queue;
 };

+struct cfq_ops
+{
+};
+
+extern struct cfq_ops opt;
+
 #endif  /* _LINUX_CFQ_IOSCHED_H */
```

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

Subject: [RFC][patch 6/11][CFQ-cgroup] Add new control layer over traditional control layer
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:36:34 GMT
View Forum Message <> Reply to Message

This patch introduces CFQ meta data (cfq_meta data).
This creates new control data layer over traditional control data (cfq_data).

The new cfq optional operations:
The "cfq_init_queue_fn" defines a function that runs when a new device is plugged, namely
new I/O queue is created.

The "cfq_exit_queue_fn" defines a function that runs when device is unplugged, namely
I/O queue is removed.

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

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index 95663f9..34894d9 100644
--- a/block/cfq-cgroup.c
+++ b/block/cfq-cgroup.c
@@ -18,6 +18,8 @@
 #define CFQ_CGROUP_SLICE_SCALE  (5)
 #define CFQ_CGROUP_MAX_IOPRIO  (8)

+static const int cfq_cgroup_slice_idle = HZ / 125;
+
 struct cfq_cgroup {
  struct cgroup_subsys_state css;
  unsigned int ioprio;
@@ -30,6 +32,52 @@ static inline struct cfq_cgroup *cgroup_to_cfq_cgroup(struct cgroup *cont)
        struct cfq_cgroup, css);
 }

+/*
+ * Add device or cgroup data functions.
+ */
+static struct cfq_meta_data *cfq_cgroup_init_meta_data(struct cfq_data *cfqd, struct
request_queue *q)
+{
+ struct cfq_meta_data *cfqmd;
+
+ cfqmd = kmalloc_node(sizeof(*cfqmd), GFP_KERNEL | __GFP_ZERO, q->node);
+ if (!cfqmd) {
+  return NULL;
+ }
+ cfqmd->elv_data = cfqd;
+
+ cfqmd->cfq_driv_d.queue = q;
+ INIT_WORK(&cfqmd->cfq_driv_d.unplug_work, cfq_kick_queue);
+ cfqmd->cfq_driv_d.last_end_request = jiffies;
+
+ init_timer(&cfqmd->cfq_driv_d.idle_slice_timer);
```

```
+ cfqmd->cfq_driv_d.idle_slice_timer.function = cfq_idle_slice_timer;
+ cfqmd->cfq_driv_d.idle_slice_timer.data = (unsigned long) cfqd;
+ cfqmd->cfq_driv_d.cfq_slice_idle = cfq_cgroup_slice_idle;
+
+ return cfqmd;
+}
+
+
+struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void *data)
+{
+ struct cfq_meta_data *cfqmd = (struct cfq_meta_data *)data;
+ struct cfq_data *cfqd = __cfq_init_cfq_data(q);
+
+ if (!cfqd)
+  return NULL;
+
+ if (!cfqmd) {
+         cfqmd = cfq_cgroup_init_meta_data(cfqd, q);
+  if (!cfqmd) {
+   kfree(cfqd);
+   return NULL;
+  }
+ }
+
+ return cfqd;
+}
+
+
 static struct cgroup_subsys_state *
 cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
@@ -50,6 +98,18 @@ cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  return &cfqc->css;
 }

+
+/*
+ * Remove device or cgroup data functions.
+ */
+static void __cfq_cgroup_exit_data(struct cfq_data *cfqd)
+{
+ struct cfq_meta_data *cfqmd = cfqd->cfqmd;
+
+ __cfq_exit_data(cfqd);
+ kfree(cfqmd);
+}
+
 static void cfq_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
 {
  kfree(cgroup_to_cfq_cgroup(cont));
@@ -154,4 +214,6 @@ struct cgroup_subsys cfq_cgroup_subsys = {


 struct cfq_ops opt = {
+ .cfq_init_queue_fn = __cfq_cgroup_init_queue,
+ .cfq_exit_queue_fn = __cfq_cgroup_exit_data,
 };
diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index 245c252..b1757bc 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -1926,7 +1926,7 @@ queue_fail:
  return 1;
 }

-static void cfq_kick_queue(struct work_struct *work)
+void cfq_kick_queue(struct work_struct *work)
 {
  struct request_queue *q =  __cfq_container_of_queue(work);
  unsigned long flags;
@@ -1939,7 +1939,7 @@ static void cfq_kick_queue(struct work_struct *work)
 /*
  * Timer running if the active_queue is currently idling inside its time slice
  */
-static void cfq_idle_slice_timer(unsigned long data)
+void cfq_idle_slice_timer(unsigned long data)
 {
  struct cfq_data *cfqd = (struct cfq_data *) data;
  struct cfq_queue *cfqq;
@@ -2002,17 +2002,17 @@ static void cfq_put_async_queues(struct cfq_data *cfqd)
   cfq_put_queue(cfqd->async_idle_cfqq);
 }

-static void cfq_exit_queue(elevator_t *e)
+void __cfq_exit_data(struct cfq_data *cfqd)
 {
- struct cfq_data *cfqd = e->elevator_data;
  struct request_queue *q = CFQ_DRV_UNIQ_DATA(cfqd).queue;

  cfq_shutdown_timer_wq(cfqd);

  spin_lock_irq(q->queue_lock);
-
- if (cfqd->active_queue)
+
+ if (cfqd->active_queue) {
```

```
   __cfq_slice_expired(cfqd, cfqd->active_queue, 0);
+ }

  while (!list_empty(&cfqd->cic_list)) {
   struct cfq_io_context *cic = list_entry(cfqd->cic_list.next,
@@ -2031,8 +2031,16 @@ static void cfq_exit_queue(elevator_t *e)
  kfree(cfqd);
 }

-static void *cfq_init_queue(struct request_queue *q)
+static void cfq_exit_queue(elevator_t *e)
 {
+ struct cfq_data *cfqd = e->elevator_data;
+
+ opt.cfq_exit_queue_fn(cfqd);
+}
+
+struct cfq_data *__cfq_init_cfq_data(struct request_queue *q)
+{
+
  struct cfq_data *cfqd;

  cfqd = kmalloc_node(sizeof(*cfqd), GFP_KERNEL | __GFP_ZERO, q->node);
@@ -2042,15 +2050,6 @@ static void *cfq_init_queue(struct request_queue *q)
  cfqd->service_tree = CFQ_RB_ROOT;
  INIT_LIST_HEAD(&cfqd->cic_list);

- CFQ_DRV_UNIQ_DATA(cfqd).queue = q;
-
- 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(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work, cfq_kick_queue);
-
- 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];
@@ -2059,7 +2058,39 @@ 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;
- CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle = cfq_slice_idle;
+
+ return cfqd;
+}
+
```

```
+#ifndef CONFIG_CGROUP_CFQ
+static struct cfq_data *__cfq_init_queue(struct request_queue *q, void *data)
+{
+ struct cfq_data *cfqd = __cfq_init_cfq_data(q);
+
+ if (!cfqd)
+  return NULL;
+
+ CFQ_DRV_UNIQ_DATA(cfqd).queue = q;
+
+ INIT_WORK(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work, cfq_kick_queue);
+
+ CFQ_DRV_UNIQ_DATA(cfqd).last_end_request = jiffies;
+
+ 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;
+ CFQ_DRV_UNIQ_DATA(cfqd).cfq_slice_idle = cfq_slice_idle;
+
+ return cfqd;
+}
+#endif
+
+static void *cfq_init_queue(struct request_queue *q)
+{
+ struct cfq_data *cfqd = NULL;
+
+ cfqd = opt.cfq_init_queue_fn(q, NULL);

  return cfqd;
 }
@@ -2235,6 +2266,8 @@ module_exit(cfq_exit);

 #ifndef CONFIG_CGROUP_CFQ
 struct cfq_ops opt = {
+ .cfq_init_queue_fn = __cfq_init_queue,
+ .cfq_exit_queue_fn = __cfq_exit_data,
 };
 #endif

diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index 9287da1..e5b41da 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -19,8 +19,11 @@ struct cfq_rb_root {
 };
 #define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }
```

```
-
+#ifdef CONFIG_CGROUP_CFQ
+#define CFQ_DRV_UNIQ_DATA(cfqd) ((cfqd)->cfqmd->cfq_driv_d)
+#else
 #define CFQ_DRV_UNIQ_DATA(cfqd) ((cfqd)->cfq_driv_d)
+#endif

 /*
  * Driver unique data
@@ -45,6 +48,17 @@ struct cfq_driver_data {
  unsigned int cfq_slice_idle;
 };

+#ifdef CONFIG_CGROUP_CFQ
+/*
+ * Per block device group queue structure
+ */
+struct cfq_meta_data {
+ struct cfq_data *elv_data;
+
+ struct cfq_driver_data cfq_driv_d;
+};
+#endif
+
 /*
  * Per block device queue structure
  */
@@ -75,9 +89,23 @@ struct cfq_data {

  struct list_head cic_list;

+#ifdef CONFIG_CGROUP_CFQ
+ struct cfq_meta_data *cfqmd;
+#else
  struct cfq_driver_data cfq_driv_d;
+#endif
 };

+#ifdef CONFIG_CGROUP_CFQ
+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_meta_data *cfqmd =
+  container_of(cfqdd, struct cfq_meta_data, cfq_driv_d);
+
+ return cfqmd->cfq_driv_d.queue;
+};
```

```
+#else
 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);
@@ -86,11 +114,23 @@ static inline struct request_queue * __cfq_container_of_queue(struct
work_struct

  return cfqd->cfq_driv_d.queue;
 };
+#endif
+
+typedef struct cfq_data *(cfq_init_queue_fn)(struct request_queue *, void *);
+typedef void (cfq_exit_queue_fn)(struct cfq_data *);

 struct cfq_ops
 {
+ cfq_init_queue_fn *cfq_init_queue_fn;
+ cfq_exit_queue_fn *cfq_exit_queue_fn;
 };

 extern struct cfq_ops opt;

+
+extern struct cfq_data *__cfq_init_cfq_data(struct request_queue *q);
+extern void cfq_kick_queue(struct work_struct *work);
+extern void cfq_idle_slice_timer(unsigned long data);
+extern void __cfq_exit_data(struct cfq_data *cfqd);
+
 #endif  /* _LINUX_CFQ_IOSCHED_H */
```

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

---

Subject: [RFC][patch 7/11][CFQ-cgroup] Control cfq_data per driver
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:37:40 GMT
View Forum Message <> Reply to Message

This patch expands cfq_meta_data to handling multi cfq_data.
This control is used rb_tree and the key is used a pointer of cfq_data.

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

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index 34894d9..55090dc 100644
--- a/block/cfq-cgroup.c
```

```
+++ b/block/cfq-cgroup.c
@@ -54,9 +54,42 @@ static struct cfq_meta_data *cfq_cgroup_init_meta_data(struct cfq_data
*cfqd, st
  cfqmd->cfq_driv_d.idle_slice_timer.data = (unsigned long) cfqd;
  cfqmd->cfq_driv_d.cfq_slice_idle = cfq_cgroup_slice_idle;

+ cfqmd->sibling_tree = RB_ROOT;
+ cfqmd->siblings = 0;
+
  return cfqmd;
 }

+static void cfq_meta_data_sibling_tree_add(struct cfq_meta_data *cfqmd,
+          struct cfq_data *cfqd)
+{
+ struct rb_node **p;
+ struct rb_node *parent = NULL;
+
+ BUG_ON(!RB_EMPTY_NODE(&cfqd->sib_node));
+
+ p = &cfqmd->sibling_tree.rb_node;
+
+ while (*p) {
+  struct cfq_data *__cfqd;
+  struct rb_node **n;
+
+  parent = *p;
+  __cfqd = rb_entry(parent, struct cfq_data, sib_node);
+
+  if (cfqd < __cfqd) {
+   n = &(*p)->rb_left;
+  } else {
+   n = &(*p)->rb_right;
+  }
+  p = n;
+ }
+
+ rb_link_node(&cfqd->sib_node, parent, p);
+ rb_insert_color(&cfqd->sib_node, &cfqmd->sibling_tree);
+ cfqmd->siblings++;
+ cfqd->cfqmd = cfqmd;
+}

 struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void *data)
 {
@@ -66,6 +99,8 @@ struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void
*data)
  if (!cfqd)
```

```
   return NULL;

+ RB_CLEAR_NODE(&cfqd->sib_node);
+
  if (!cfqmd) {
          cfqmd = cfq_cgroup_init_meta_data(cfqd, q);
   if (!cfqmd) {
@@ -73,6 +108,7 @@ struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void
*data)
    return NULL;
   }
  }
+ cfq_meta_data_sibling_tree_add(cfqmd, cfqd);

  return cfqd;
 }
@@ -102,11 +138,35 @@ cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 /*
  * Remove device or cgroup data functions.
  */
+static void cfq_cgroup_erase_meta_data_siblings(struct cfq_meta_data *cfqmd, struct cfq_data
*cfqd)
+{
+ rb_erase(&cfqd->sib_node, &cfqmd->sibling_tree);
+ cfqmd->siblings--;
+}
+
+static void cfq_exit_device_group(struct cfq_meta_data *cfqmd)
+{
+ struct rb_node *p, *n;
+ struct cfq_data *cfqd;
+
+ p =  rb_first(&cfqmd->sibling_tree);
+
+ while (p) {
+  n = rb_next(p);
+  cfqd = rb_entry(p, struct cfq_data, sib_node);
+
+  cfq_cgroup_erase_meta_data_siblings(cfqmd, cfqd);
+  __cfq_exit_data(cfqd);
+
+  p = n;
+ }
+}
+
 static void __cfq_cgroup_exit_data(struct cfq_data *cfqd)
 {
  struct cfq_meta_data *cfqmd = cfqd->cfqmd;
```

```
- __cfq_exit_data(cfqd);
+ cfq_exit_device_group(cfqmd);
  kfree(cfqmd);
 }

diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index e5b41da..c8879a7 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -54,6 +54,10 @@ struct cfq_driver_data {
  */
 struct cfq_meta_data {
  struct cfq_data *elv_data;
+
+ /* device siblings */
+ struct rb_root sibling_tree;
+ unsigned int siblings;

  struct cfq_driver_data cfq_driv_d;
 };
@@ -91,6 +95,9 @@ struct cfq_data {

 #ifdef CONFIG_CGROUP_CFQ
  struct cfq_meta_data *cfqmd;
+ /* sibling_tree member for cfq_meta_data */
+ struct rb_node sib_node;
+
 #else
  struct cfq_driver_data cfq_driv_d;
 #endif
```

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

---

Subject: [RFC][patch 8/11][CFQ-cgroup] Control cfq_data per cgroup
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:38:52 GMT
View Forum Message <> Reply to Message

This patch expands cfq data to handling multi cfq_data in group.
This control is used rb_tree and the key is used a pointer of cfq_meta_data.

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

diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c

```
index 55090dc..4110ab7 100644
--- a/block/cfq-cgroup.c
+++ b/block/cfq-cgroup.c
@@ -23,6 +23,9 @@ static const int cfq_cgroup_slice_idle = HZ / 125;
 struct cfq_cgroup {
  struct cgroup_subsys_state css;
  unsigned int ioprio;
+
+ struct rb_root sibling_tree;
+ unsigned int siblings;
 };


@@ -35,6 +38,8 @@ static inline struct cfq_cgroup *cgroup_to_cfq_cgroup(struct cgroup *cont)
 /*
  * Add device or cgroup data functions.
  */
+struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void *data);
+
 static struct cfq_meta_data *cfq_cgroup_init_meta_data(struct cfq_data *cfqd, struct
request_queue *q)
 {
  struct cfq_meta_data *cfqmd;
@@ -90,16 +95,75 @@ static void cfq_meta_data_sibling_tree_add(struct cfq_meta_data
*cfqmd,
  cfqmd->siblings++;
  cfqd->cfqmd = cfqmd;
 }
-
+
+static void cfq_cgroup_sibling_tree_add(struct cfq_cgroup *cfqc,
+     struct cfq_data *cfqd)
+{
+ struct rb_node **p;
+ struct rb_node *parent = NULL;
+
+ BUG_ON(!RB_EMPTY_NODE(&cfqd->group_node));
+
+ p = &cfqc->sibling_tree.rb_node;
+
+ while (*p) {
+  struct cfq_data *__cfqd;
+  struct rb_node **n;
+
+  parent = *p;
+  __cfqd = rb_entry(parent, struct cfq_data, group_node);
+
+  if (cfqd->cfqmd < __cfqd->cfqmd) {
```

```
+   n = &(*p)->rb_left;
+  } else {
+   n = &(*p)->rb_right;
+  }
+  p = n;
+ }
+
+ rb_link_node(&cfqd->group_node, parent, p);
+ rb_insert_color(&cfqd->group_node, &cfqc->sibling_tree);
+ cfqc->siblings++;
+ cfqd->cfqc = cfqc;
+}
+
+static void *cfq_cgroup_init_cfq_data(struct cfq_cgroup *cfqc, struct cfq_data *cfqd)
+{
+ struct cgroup *child;
+
+ /* setting cfq_data for cfq_cgroup */
+ if (!cfqc) {
+  cfqc = cgroup_to_cfq_cgroup(get_root_subsys(&cfq_cgroup_subsys));
+  cfq_cgroup_sibling_tree_add(cfqc, cfqd);
+ } else {
+  struct cfq_data *__cfqd;
+  __cfqd = __cfq_cgroup_init_queue(CFQ_DRV_UNIQ_DATA(cfqd).queue, cfqd->cfqmd);
+  if (!__cfqd)
+   return NULL;
+  cfq_cgroup_sibling_tree_add(cfqc, __cfqd);
+ }
+
+ /* check and create cfq_data for children */
+ if (cfqc->css.cgroup)
+  list_for_each_entry(child, &cfqc->css.cgroup->children, children){
+   cfq_cgroup_init_cfq_data(cgroup_to_cfq_cgroup(child), cfqd);
+  }
+
+ return cfqc;
+}
+
+
 struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void *data)
 {
  struct cfq_meta_data *cfqmd = (struct cfq_meta_data *)data;
  struct cfq_data *cfqd = __cfq_init_cfq_data(q);
+ int root = 0;

  if (!cfqd)
   return NULL;
```

```
  RB_CLEAR_NODE(&cfqd->sib_node);
+   RB_CLEAR_NODE(&cfqd->group_node);

  if (!cfqmd) {
          cfqmd = cfq_cgroup_init_meta_data(cfqd, q);
@@ -107,12 +171,35 @@ struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q,
void *data)
    kfree(cfqd);
    return NULL;
   }
+  root = 1;
  }
  cfq_meta_data_sibling_tree_add(cfqmd, cfqd);

+ if (root)
+  cfq_cgroup_init_cfq_data(NULL, cfqd);
+
  return cfqd;
 }

+static void *cfq_cgroup_init_cgroup(struct cfq_cgroup *cfqc, struct cgroup *parent)
+{
+ struct rb_node *p;
+ if (parent) {
+  struct cfq_cgroup *cfqc_p = cgroup_to_cfq_cgroup(parent);
+
+  p = rb_first(&cfqc_p->sibling_tree);
+  while (p != NULL) {
+   struct cfq_data *__cfqd;
+   __cfqd = rb_entry(p, struct cfq_data, group_node);
+
+   cfq_cgroup_init_cfq_data(cfqc, __cfqd);
+
+   p = rb_next(p);
+  }
+ }
+
+ return cfqc;
+}

 static struct cgroup_subsys_state *
 cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
@@ -130,6 +217,11 @@ cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
   return ERR_PTR(-ENOMEM);

  cfqc->ioprio = 3;
+ cfqc->sibling_tree = RB_ROOT;
+ cfqc->siblings = 0;
```

```
+
+ if (!cfq_cgroup_init_cgroup(cfqc, cont->parent))
+  return ERR_PTR(-ENOMEM);

  return &cfqc->css;
 }
@@ -144,6 +236,12 @@ static void cfq_cgroup_erase_meta_data_siblings(struct cfq_meta_data
*cfqmd, str
  cfqmd->siblings--;
 }

+static void cfq_cgroup_erase_cgroup_siblings(struct cfq_cgroup *cfqc, struct cfq_data *cfqd)
+{
+ rb_erase(&cfqd->group_node, &cfqc->sibling_tree);
+ cfqc->siblings--;
+}
+
 static void cfq_exit_device_group(struct cfq_meta_data *cfqmd)
 {
  struct rb_node *p, *n;
@@ -156,6 +254,7 @@ static void cfq_exit_device_group(struct cfq_meta_data *cfqmd)
   cfqd = rb_entry(p, struct cfq_data, sib_node);

   cfq_cgroup_erase_meta_data_siblings(cfqmd, cfqd);
+  cfq_cgroup_erase_cgroup_siblings(cfqd->cfqc, cfqd);
   __cfq_exit_data(cfqd);

   p = n;
@@ -170,8 +269,28 @@ static void __cfq_cgroup_exit_data(struct cfq_data *cfqd)
  kfree(cfqmd);
 }

+static void cfq_exit_cgroup(struct cfq_cgroup *cfqc)
+{
+ struct rb_node *p, *n;
+ struct cfq_data *cfqd;
+
+ p =  rb_first(&cfqc->sibling_tree);
+
+ while (p) {
+  n = rb_next(p);
+  cfqd = rb_entry(p, struct cfq_data, group_node);
+
+  cfq_cgroup_erase_meta_data_siblings(cfqd->cfqmd, cfqd);
+  cfq_cgroup_erase_cgroup_siblings(cfqc, cfqd);
+  __cfq_exit_data(cfqd);
+
+  p = n;
```

```
+ }
+}
+
 static void cfq_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
 {
+ cfq_exit_cgroup(cgroup_to_cfq_cgroup(cont));
  kfree(cgroup_to_cfq_cgroup(cont));
 }

diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index c8879a7..93256ce 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -94,10 +94,16 @@ struct cfq_data {
  struct list_head cic_list;

 #ifdef CONFIG_CGROUP_CFQ
+ /* device unique attribute */
  struct cfq_meta_data *cfqmd;
  /* sibling_tree member for cfq_meta_data */
  struct rb_node sib_node;

+ /* cfq_cgroup attribute */
+ struct cfq_cgroup *cfqc;
+ /* group_tree member for cfq_cgroup */
+ struct rb_node group_node;
+
 #else
  struct cfq_driver_data cfq_driv_d;
 #endif
diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index 785a01c..6423951 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -356,6 +356,7 @@ struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
 void cgroup_iter_end(struct cgroup *cgrp, struct cgroup_iter *it);
 int cgroup_scan_tasks(struct cgroup_scanner *scan);
 int cgroup_attach_task(struct cgroup *, struct task_struct *);
+struct cgroup* get_root_subsys(struct cgroup_subsys *css);

 #else /* !CONFIG_CGROUPS */

diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index e8e8ec4..1ef1de7 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -1290,6 +1290,12 @@ static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
  return ret;
```

```
 }

+
+struct cgroup* get_root_subsys(struct cgroup_subsys *css)
+{
+ return &css->root->top_cgroup;
+}
+
 /* The various types of files and directories in a cgroup file system */
 enum cgroup_filetype {
  FILE_ROOT,
```

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

---

Subject: [RFC][patch 9/11][CFQ-cgroup] Search cfq_data when not connected
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:40:04 GMT
View Forum Message <> Reply to Message

This patch is possible to select a cfq_data corresponding group with task.
This is used when merge, merge check, queue check and queue setting.

The new cfq optional operations:
The "cfq_search_data_fn" defines a function that selects a correct cfq_data when
cfq_queue and requests are not connected yet.

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

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index 4110ab7..568e433 100644
--- a/block/cfq-cgroup.c
+++ b/block/cfq-cgroup.c
@@ -35,6 +35,12 @@ static inline struct cfq_cgroup *cgroup_to_cfq_cgroup(struct cgroup *cont)
       struct cfq_cgroup, css);
 }

+static inline struct cfq_cgroup *task_to_cfq_cgroup(struct task_struct *tsk)
+{
+ return container_of(task_subsys_state(tsk, cfq_cgroup_subsys_id),
+       struct cfq_cgroup, css);
+}
+
 /*
  * Add device or cgroup data functions.
  */
```

```
@@ -392,6 +398,32 @@ struct cgroup_subsys cfq_cgroup_subsys = {
 };


+struct cfq_data *cfq_cgroup_search_data(void *data,
+     struct task_struct *tsk)
+{
+ struct cfq_data *cfqd = (struct cfq_data *)data;
+ struct cfq_meta_data *cfqmd = cfqd->cfqmd;
+ struct cfq_cgroup *cont = task_to_cfq_cgroup(tsk);
+ struct rb_node *p = cont->sibling_tree.rb_node;
+
+ while (p) {
+  struct cfq_data *__cfqd;
+  __cfqd = rb_entry(p, struct cfq_data, group_node);
+
+
+  if (cfqmd < __cfqd->cfqmd) {
+   p = p->rb_left;
+  } else if (cfqmd > __cfqd->cfqmd) {
+   p = p->rb_right;
+  } else {
+   return __cfqd;
+  }
+ }
+
+ return NULL;
+}
+
+
 struct cfq_ops opt = {
  .cfq_init_queue_fn = __cfq_cgroup_init_queue,
  .cfq_exit_queue_fn = __cfq_cgroup_exit_data,
diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index b1757bc..3aa320a 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -623,6 +623,9 @@ static int cfq_merge(struct request_queue *q, struct request **req,
  struct cfq_data *cfqd = q->elevator->elevator_data;
  struct request *__rq;

+ if (opt.cfq_search_data_fn)
+  cfqd = opt.cfq_search_data_fn(cfqd, current);
+
  __rq = cfq_find_rq_fmerge(cfqd, bio);
  if (__rq && elv_rq_merge_ok(__rq, bio)) {
   *req = __rq;
@@ -663,6 +666,9 @@ static int cfq_allow_merge(struct request_queue *q, struct request *rq,
```

```
  struct cfq_io_context *cic;
  struct cfq_queue *cfqq;

+ if (opt.cfq_search_data_fn)
+  cfqd = opt.cfq_search_data_fn(cfqd, current);
+
  /*
   * Disallow merge of a sync bio into an async request.
   */
@@ -1832,6 +1838,9 @@ static int cfq_may_queue(struct request_queue *q, int rw)
  struct cfq_io_context *cic;
  struct cfq_queue *cfqq;

+ if (opt.cfq_search_data_fn)
+  cfqd = opt.cfq_search_data_fn(cfqd, current);
+
  /*
   * don't force setup of a queue from here, as a call to may_queue
   * does not necessarily imply that a request actually will be queued.
@@ -1888,6 +1897,9 @@ cfq_set_request(struct request_queue *q, struct request *rq, gfp_t
gfp_mask)
  struct cfq_queue *cfqq;
  unsigned long flags;

+ if (opt.cfq_search_data_fn)
+  cfqd = opt.cfq_search_data_fn(cfqd, current);
+
  might_sleep_if(gfp_mask & __GFP_WAIT);

  cic = cfq_get_io_context(cfqd, gfp_mask);
diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index 93256ce..a382953 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -131,11 +131,13 @@ static inline struct request_queue * __cfq_container_of_queue(struct
work_struct

 typedef struct cfq_data *(cfq_init_queue_fn)(struct request_queue *, void *);
 typedef void (cfq_exit_queue_fn)(struct cfq_data *);
+typedef struct cfq_data *(cfq_search_data_fn)(void *, struct task_struct *);

 struct cfq_ops
 {
  cfq_init_queue_fn *cfq_init_queue_fn;
  cfq_exit_queue_fn *cfq_exit_queue_fn;
+ cfq_search_data_fn *cfq_search_data_fn;
 };
```

```
 extern struct cfq_ops opt;
```

_____

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

---

Subject: [RFC][patch 10/11][CFQ-cgroup] Control service tree: Main functions
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:41:27 GMT
View Forum Message <> Reply to Message

This patch introduced to control cfq_data.
Its algorithm is similar to one when CFQ synchronous I/O.

The new cfq optional operations:
The "cfq_dispatch_requests_fn" defines a function which is implemented
request dispatching algorithm.
This becomes main function for fairness.

The "cfq_completed_request_after_fn" defines a function which winds up I/O's
affairs.

The "cfq_active_check_fn" defines a function which make sure whether selecting cfq_data is
equal to active cfq_data.

The "cfq_empty_fn" defines a function which check whether active data exists.

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

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index 568e433..f040c98 100644
--- a/block/cfq-cgroup.c
+++ b/block/cfq-cgroup.c
@@ -15,9 +15,35 @@
 #include <linux/cgroup.h>
 #include <linux/cfq-iosched.h>

+
 #define CFQ_CGROUP_SLICE_SCALE  (5)
 #define CFQ_CGROUP_MAX_IOPRIO  (8)

+static const int cfq_cgroup_slice = HZ / 10;
+
+enum cfqd_state_flags {
+ CFQ_CFQD_FLAG_on_rr = 0, /* on round-robin busy list */
+ CFQ_CFQD_FLAG_slice_new, /* no requests dispatched in slice */
+};
```

```
+
+#define CFQ_CFQD_FNS(name)      \
+static inline void cfq_mark_cfqd_##name(struct cfq_data *cfqd)  \
+{       \
+ (cfqd)->flags |= (1 << CFQ_CFQD_FLAG_##name);   \
+}       \
+static inline void cfq_clear_cfqd_##name(struct cfq_data *cfqd) \
+{       \
+ (cfqd)->flags &= ~(1 << CFQ_CFQD_FLAG_##name);   \
+}       \
+static inline int cfq_cfqd_##name(const struct cfq_data *cfqd)  \
+{       \
+ return ((cfqd)->flags & (1 << CFQ_CFQD_FLAG_##name)) != 0; \
+}
+
+CFQ_CFQD_FNS(on_rr);
+CFQ_CFQD_FNS(slice_new);
+#undef CFQ_CFQD_FNS
+
 static const int cfq_cgroup_slice_idle = HZ / 125;

 struct cfq_cgroup {
@@ -45,6 +71,7 @@ static inline struct cfq_cgroup *task_to_cfq_cgroup(struct task_struct *tsk)
  * Add device or cgroup data functions.
  */
 struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void *data);
+static void cfq_cgroup_idle_slice_timer(unsigned long data);

 static struct cfq_meta_data *cfq_cgroup_init_meta_data(struct cfq_data *cfqd, struct
request_queue *q)
 {
@@ -61,13 +88,18 @@ static struct cfq_meta_data *cfq_cgroup_init_meta_data(struct cfq_data
*cfqd, st
  cfqmd->cfq_driv_d.last_end_request = jiffies;

  init_timer(&cfqmd->cfq_driv_d.idle_slice_timer);
- cfqmd->cfq_driv_d.idle_slice_timer.function = cfq_idle_slice_timer;
- cfqmd->cfq_driv_d.idle_slice_timer.data = (unsigned long) cfqd;
+ cfqmd->cfq_driv_d.idle_slice_timer.function = cfq_cgroup_idle_slice_timer;
+ cfqmd->cfq_driv_d.idle_slice_timer.data = (unsigned long) cfqmd;
  cfqmd->cfq_driv_d.cfq_slice_idle = cfq_cgroup_slice_idle;

  cfqmd->sibling_tree = RB_ROOT;
  cfqmd->siblings = 0;

+ cfqmd->service_tree = CFQ_RB_ROOT;
+ cfqmd->busy_data = 0;
+
```

```
+ cfqmd->cfq_slice = cfq_cgroup_slice;
+
  return cfqmd;
 }

@@ -170,6 +202,8 @@ struct cfq_data *__cfq_cgroup_init_queue(struct request_queue *q, void
*data)

  RB_CLEAR_NODE(&cfqd->sib_node);
    RB_CLEAR_NODE(&cfqd->group_node);
+   RB_CLEAR_NODE(&cfqd->rb_node);
+ cfqd->rb_key = 0;

  if (!cfqmd) {
          cfqmd = cfq_cgroup_init_meta_data(cfqd, q);
@@ -424,7 +458,295 @@ struct cfq_data *cfq_cgroup_search_data(void *data,
 }


+/*
+ *  service tree control.
+ */
+static inline int cfq_cgroup_slice_used(struct cfq_data *cfqd)
+{
+   if (cfq_cfqd_slice_new(cfqd))
+  return 0;
+   if (time_before(jiffies, cfqd->slice_end))
+  return 0;
+
+ return 1;
+}
+
+
+static inline int cfq_cgroup_prio_slice(struct cfq_data *cfqd,
+     unsigned short prio)
+{
+ const int base_slice = cfqd->cfqmd->cfq_slice;
+
+ WARN_ON(prio >= IOPRIO_BE_NR);
+
+ return base_slice + (base_slice/CFQ_CGROUP_SLICE_SCALE *
+        (CFQ_CGROUP_MAX_IOPRIO / 2 - prio));
+}
+
+static inline void
+cfq_cgroup_set_prio_slice(struct cfq_data *cfqd)
+{
+ cfqd->slice_end = cfq_cgroup_prio_slice(cfqd, cfqd->cfqc->ioprio) + jiffies;
```

```
+}
+
+static unsigned long cfq_cgroup_slice_offset(struct cfq_data *cfqd)
+{
+ return (cfqd->cfqmd->busy_data - 1) *
+  (cfq_cgroup_prio_slice(cfqd, 0) -
+   cfq_cgroup_prio_slice(cfqd, cfqd->cfqc->ioprio));
+}
+
+static void cfq_cgroup_service_tree_add(struct cfq_data *cfqd,
+     int add_front)
+{
+ struct rb_node **p, *parent;
+ struct cfq_data *__cfqd;
+ struct cfq_meta_data *cfqmd = cfqd->cfqmd;
+ unsigned long rb_key;
+ int left;
+
+ if (!add_front) {
+  rb_key = cfq_cgroup_slice_offset(cfqd) + jiffies;
+  rb_key += cfqd->slice_resid;
+  cfqd->slice_resid = 0;
+ } else
+  rb_key = 0;
+
+ if (!RB_EMPTY_NODE(&cfqd->rb_node)) {
+  if (rb_key == cfqd->rb_key)
+   return;
+  cfq_rb_erase(&cfqd->rb_node, &cfqmd->service_tree);
+ }
+
+ left = 1;
+ parent = NULL;
+ p = &cfqmd->service_tree.rb.rb_node;
+ while (*p) {
+  struct rb_node **n;
+
+  parent = *p;
+  __cfqd = rb_entry(parent, struct cfq_data, rb_node);
+
+  if (rb_key < __cfqd->rb_key)
+   n = &(*p)->rb_left;
+  else
+   n = &(*p)->rb_right;
+
+  if (n == &(*p)->rb_right)
+   left = 0;
+
```

```
+  p = n;
+ }
+
+ if (left)
+  cfqmd->service_tree.left = &cfqd->rb_node;
+
+ cfqd->rb_key = rb_key;
+ rb_link_node(&cfqd->rb_node, parent, p);
+ rb_insert_color(&cfqd->rb_node, &cfqmd->service_tree.rb);
+}
+
+
+static void
+__cfq_cgroup_slice_expired(struct cfq_meta_data *cfqmd,struct cfq_data *cfqd,
+      int timed_out)
+{
+ if (timed_out && !cfq_cfqd_slice_new(cfqd))
+  cfqd->slice_resid = cfqd->slice_end - jiffies;
+
+ if (cfq_cfqd_on_rr(cfqd)) {
+  cfq_cgroup_service_tree_add(cfqd,0);
+ }
+
+ if (cfqd == cfqmd->active_data) {
+  cfqmd->active_data = NULL;
+ }
+}
+
+static inline void
+cfq_cgroup_slice_expired(struct cfq_meta_data *cfqmd, int timed_out)
+{
+ struct cfq_data *cfqd = cfqmd->active_data;
+
+ if (cfqd) {
+  cfq_slice_expired(cfqd, 1);
+  __cfq_cgroup_slice_expired(cfqmd, cfqd, timed_out);
+ }
+}
+
+static struct cfq_data *cfq_cgroup_rb_first(struct cfq_rb_root *root)
+{
+ if (!root->left)
+  root->left = rb_first(&root->rb);
+
+ if (root->left)
+  return rb_entry(root->left, struct cfq_data, rb_node);
+
+ return NULL;
```

```
+}
+
+static struct cfq_data *cfq_cgroup_get_next_data(struct cfq_meta_data *cfqmd)
+{
+ if (RB_EMPTY_ROOT(&cfqmd->service_tree.rb))
+  return NULL;
+
+ return cfq_cgroup_rb_first(&cfqmd->service_tree);
+}
+
+static void
+__cfq_cgroup_set_active_data(struct cfq_meta_data *cfqmd,
+        struct cfq_data *cfqd)
+{
+ if (cfqd) {
+  cfqd->slice_end = 0;
+  cfq_mark_cfqd_slice_new(cfqd);
+ }
+
+ cfqmd->active_data = cfqd;
+}
+
+static struct cfq_data *cfq_cgroup_set_active_data(struct cfq_meta_data *cfqmd)
+{
+ struct cfq_data *cfqd;
+
+ cfqd = cfq_cgroup_get_next_data(cfqmd);
+ __cfq_cgroup_set_active_data(cfqmd , cfqd);
+
+ return cfqd;
+}
+
+struct cfq_data *cfq_cgroup_select_data(struct cfq_meta_data *cfqmd)
+{
+ struct cfq_data *cfqd;
+
+ cfqd = cfqmd->active_data;
+ if (!cfqd)
+  goto new_data;
+
+ if (cfq_cgroup_slice_used(cfqd))
+  goto expire;
+
+ if (!RB_EMPTY_ROOT(&cfqd->service_tree.rb))
+  goto keep_data;
+
+ if (wait_request_checker(cfqd))
+  goto keep_data;
```

```
+
+expire:
+ cfq_cgroup_slice_expired(cfqmd, 0);
+new_data:
+ cfqd = cfq_cgroup_set_active_data(cfqmd);
+keep_data:
+ return cfqd;
+}
+
+int cfq_cgroup_forced_dispatch(struct cfq_data *cfqd)
+{
+ struct cfq_meta_data *cfqmd = cfqd->cfqmd;
+ int dispatched = 0;
+
+ while ((cfqd = cfq_cgroup_rb_first(&cfqmd->service_tree)) != NULL)
+  dispatched += cfq_forced_dispatch(cfqd);
+
+ cfq_cgroup_slice_expired(cfqmd, 0);
+
+ BUG_ON(cfqmd->busy_data);
+
+ return dispatched;
+}
+
+int cfq_cgroup_dispatch_requests(struct cfq_data *cfqd, int force)
+{
+ struct cfq_meta_data *cfqmd = cfqd->cfqmd;
+ int dispatched;
+
+
+ if (!cfqmd->busy_data)
+  return 0;
+
+ if (unlikely(force))
+  return cfq_cgroup_forced_dispatch(cfqd);
+
+ dispatched = 0;
+ cfqd = cfq_cgroup_select_data(cfqmd);
+
+ if (cfqd)
+  dispatched = cfq_queue_dispatch_requests(cfqd, force);
+
+ return dispatched;
+}
+
+/*
+ * Timer running if the active_queue is currently idling inside its time slice
+ */
```

```
+static void cfq_cgroup_idle_slice_timer(unsigned long data)
+{
+ struct cfq_meta_data *cfqmd = (struct cfq_meta_data *) data;
+ struct cfq_data *cfqd = cfqmd->elv_data;
+ int timed_out = 1;
+ unsigned long flags;
+
+
+ spin_lock_irqsave(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
+
+ cfqd = cfqmd->active_data;
+ if (cfqd) {
+  timed_out = 0;
+
+  if (cfq_cgroup_slice_used(cfqd))
+   goto expire_cgroup;
+
+  if (!cfqmd->busy_data)
+   goto out_cont;
+
+  if (__cfq_idle_slice_timer(cfqd))
+   goto out_cont;
+  else
+   goto out_kick;
+
+ }
+expire_cgroup:
+ cfq_cgroup_slice_expired(cfqmd, timed_out);
+out_kick:
+ cfq_schedule_dispatch(cfqmd->elv_data);
+out_cont:
+ spin_unlock_irqrestore(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
+}
+
+int cfq_cgroup_completed_request_after(struct cfq_data *cfqd)
+{
+ if (cfqd->cfqmd->active_data == cfqd) {
+  if (cfq_cfqd_slice_new(cfqd)) {
+   cfq_cgroup_set_prio_slice(cfqd);
+   cfq_clear_cfqd_slice_new(cfqd);
+
+  }
+  if (cfq_cgroup_slice_used(cfqd)) {
+   cfq_cgroup_slice_expired(cfqd->cfqmd, 1);
+   return 0;
+  }
+  return 1;
+ }
```

```
+
+ return 0;
+}
+
+static int cfq_cgroup_queue_empty(struct cfq_data *cfqd)
+{
+ return !cfqd->cfqmd->busy_data;
+}
+
+static int cfq_cgroup_active_data_check(struct cfq_data *cfqd)
+{
+ return (cfqd->cfqmd->active_data == cfqd);
+}
+
 struct cfq_ops opt = {
  .cfq_init_queue_fn = __cfq_cgroup_init_queue,
  .cfq_exit_queue_fn = __cfq_cgroup_exit_data,
+ .cfq_search_data_fn = cfq_cgroup_search_data,
+ .cfq_dispatch_requests_fn = cfq_cgroup_dispatch_requests,
+ .cfq_completed_request_after_fn = cfq_cgroup_completed_request_after,
+ .cfq_empty_fn = cfq_cgroup_queue_empty,
+ .cfq_active_check_fn = cfq_cgroup_active_data_check,
 };
diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index 3aa320a..505e425 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -174,19 +174,25 @@ static inline int cfq_bio_sync(struct bio *bio)
  * scheduler run of queue, if there are requests pending and no one in the
  * driver that will restart queueing
  */
-static inline void cfq_schedule_dispatch(struct cfq_data *cfqd)
+#ifndef CONFIG_CGROUP_CFQ
+static int __cfq_queue_empty(struct cfq_data *cfqd)
 {
- if (cfqd->busy_queues)
-  kblockd_schedule_work(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work);
+ return !cfqd->busy_queues;
 }
+#endif

 static int cfq_queue_empty(struct request_queue *q)
 {
  struct cfq_data *cfqd = q->elevator->elevator_data;

- return !cfqd->busy_queues;
+ return opt.cfq_empty_fn(cfqd);
 }
```

```
+inline void cfq_schedule_dispatch(struct cfq_data *cfqd)
+{
+ if (!opt.cfq_empty_fn(cfqd))
+  kblockd_schedule_work(&CFQ_DRV_UNIQ_DATA(cfqd).unplug_work);
+}
 /*
  * Scale schedule slice based on io priority. Use the sync time slice only
  * if a queue is marked sync and has sync io queued. A sync queue with async
@@ -338,7 +344,7 @@ static struct cfq_queue *cfq_rb_first(struct cfq_rb_root *root)
  return NULL;
 }

-static void cfq_rb_erase(struct rb_node *n, struct cfq_rb_root *root)
+void cfq_rb_erase(struct rb_node *n, struct cfq_rb_root *root)
 {
  if (root->left == n)
   root->left = NULL;
@@ -734,7 +740,7 @@ __cfq_slice_expired(struct cfq_data *cfqd, struct cfq_queue *cfqq,
  }
 }

-static inline void cfq_slice_expired(struct cfq_data *cfqd, int timed_out)
+inline void cfq_slice_expired(struct cfq_data *cfqd, int timed_out)
 {
  struct cfq_queue *cfqq = cfqd->active_queue;

@@ -847,8 +853,8 @@ static void cfq_arm_slice_timer(struct cfq_data *cfqd)
  */
 static void cfq_dispatch_insert(struct request_queue *q, struct request *rq)
 {
- struct cfq_data *cfqd = q->elevator->elevator_data;
  struct cfq_queue *cfqq = RQ_CFQQ(rq);
+ struct cfq_data *cfqd = cfqq->cfqd;

  cfq_remove_request(rq);
  cfqq->dispatched++;
@@ -894,6 +900,17 @@ cfq_prio_to_maxrq(struct cfq_data *cfqd, struct cfq_queue *cfqq)
  return 2 * (base_rq + base_rq * (CFQ_PRIO_LISTS - 1 - cfqq->ioprio));
 }

+
+int wait_request_checker(struct cfq_data *cfqd)
+{
+ struct cfq_queue *cfqq = cfqd->active_queue;
+ if (cfqq)
+  return (timer_pending(&CFQ_DRV_UNIQ_DATA(cfqd).idle_slice_timer) ||
+   (cfqq->dispatched && cfq_cfqq_idle_window(cfqq)));
```

```
+ else
+  return 0;
+}
+
 /*
  * Select a queue for service. If we have a current active queue,
  * check whether to continue servicing it, or retrieve and set a new one.
@@ -1008,7 +1025,7 @@ static int __cfq_forced_dispatch_cfqq(struct cfq_queue *cfqq)
  * Drain our current requests. Used for barriers and when switching
  * io schedulers on-the-fly.
  */
-static int cfq_forced_dispatch(struct cfq_data *cfqd)
+int cfq_forced_dispatch(struct cfq_data *cfqd)
 {
  struct cfq_queue *cfqq;
  int dispatched = 0;
@@ -1023,9 +1040,8 @@ static int cfq_forced_dispatch(struct cfq_data *cfqd)
  return dispatched;
 }

-static int cfq_dispatch_requests(struct request_queue *q, int force)
+int cfq_queue_dispatch_requests(struct cfq_data *cfqd, int force)
 {
- struct cfq_data *cfqd = q->elevator->elevator_data;
  struct cfq_queue *cfqq;
  int dispatched;

@@ -1063,6 +1079,15 @@ static int cfq_dispatch_requests(struct request_queue *q, int force)
  return dispatched;
 }

+static int cfq_dispatch_requests(struct request_queue *q, int force)
+{
+ struct cfq_data *cfqd = q->elevator->elevator_data;
+
+ if (opt.cfq_dispatch_requests_fn)
+  return opt.cfq_dispatch_requests_fn(cfqd, force);
+ return 0;
+}
+
 /*
  * task holds one reference to the queue, dropped when task exits. each rq
  * in-flight on this queue also holds a reference, dropped when rq is freed.
@@ -1635,6 +1660,12 @@ cfq_should_preempt(struct cfq_data *cfqd, struct cfq_queue
*new_cfqq,
      struct request *rq)
 {
  struct cfq_queue *cfqq;
```

```
+ int flag=1;
+
+ if (opt.cfq_active_check_fn)
+  flag = opt.cfq_active_check_fn(cfqd);
+ if (!flag)
+  return 0;

  cfqq = cfqd->active_queue;
  if (!cfqq)
@@ -1705,6 +1736,7 @@ cfq_rq_enqueued(struct cfq_data *cfqd, struct cfq_queue *cfqq,
   struct request *rq)
 {
  struct cfq_io_context *cic = RQ_CIC(rq);
+ int flag=1;

  if (rq_is_meta(rq))
   cfqq->meta_pending++;
@@ -1715,7 +1747,10 @@ cfq_rq_enqueued(struct cfq_data *cfqd, struct cfq_queue *cfqq,

  cic->last_request_pos = rq->sector + rq->nr_sectors;

- if (cfqq == cfqd->active_queue) {
+ if (opt.cfq_active_check_fn)
+  flag = opt.cfq_active_check_fn(cfqd);
+
+ if ((flag) && (cfqq == cfqd->active_queue)) {
   /*
    * if we are waiting for a request for this queue, let it rip
    * immediately and flag that we must not expire this queue
@@ -1740,9 +1775,9 @@ cfq_rq_enqueued(struct cfq_data *cfqd, struct cfq_queue *cfqq,

 static void cfq_insert_request(struct request_queue *q, struct request *rq)
 {
- struct cfq_data *cfqd = q->elevator->elevator_data;
  struct cfq_queue *cfqq = RQ_CFQQ(rq);
-
+ struct cfq_data *cfqd = cfqq->cfqd;
+
  cfq_init_prio_data(cfqq, RQ_CIC(rq)->ioc);

  cfq_add_rq_rb(rq);
@@ -1758,6 +1793,7 @@ static void cfq_completed_request(struct request_queue *q, struct
request *rq)
  struct cfq_data *cfqd = cfqq->cfqd;
  const int sync = rq_is_sync(rq);
  unsigned long now;
+ int flag=1;
```

```
  now = jiffies;

@@ -1779,7 +1815,10 @@ static void cfq_completed_request(struct request_queue *q, struct
request *rq)
   * If this is the active queue, check if it needs to be expired,
   * or if we want to idle in case it has no pending requests.
   */
- if (cfqd->active_queue == cfqq) {
+ if (opt.cfq_completed_request_after_fn)
+  flag = opt.cfq_completed_request_after_fn(cfqd);
+
+ if ((flag) && (cfqd->active_queue == cfqq)) {
   if (cfq_cfqq_slice_new(cfqq)) {
    cfq_set_prio_slice(cfqd, cfqq);
    cfq_clear_cfqq_slice_new(cfqq);
@@ -1951,15 +1990,11 @@ void cfq_kick_queue(struct work_struct *work)
 /*
  * Timer running if the active_queue is currently idling inside its time slice
  */
-void cfq_idle_slice_timer(unsigned long data)
+inline int __cfq_idle_slice_timer(struct cfq_data *cfqd)
 {
- struct cfq_data *cfqd = (struct cfq_data *) data;
  struct cfq_queue *cfqq;
- unsigned long flags;
  int timed_out = 1;

- spin_lock_irqsave(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
-
  cfqq = cfqd->active_queue;
  if (cfqq) {
   timed_out = 0;
@@ -1989,7 +2024,21 @@ expire:
  cfq_slice_expired(cfqd, timed_out);
 out_kick:
  cfq_schedule_dispatch(cfqd);
+ return 1;
 out_cont:
+ return 0;
+}
+
+
+void cfq_idle_slice_timer(unsigned long data)
+{
+ struct cfq_data *cfqd = (struct cfq_data *) data;
+ unsigned long flags;
+
+ spin_lock_irqsave(CFQ_DRV_UNIQ_DATA(cfqd).queue->queue_lock, flags);
```

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

@@ -2280,6 +2329,8 @@ module_exit(cfq_exit);
 struct cfq_ops opt = {
  .cfq_init_queue_fn = __cfq_init_queue,
  .cfq_exit_queue_fn = __cfq_exit_data,
+ .cfq_dispatch_requests_fn = cfq_queue_dispatch_requests,
+ .cfq_empty_fn = __cfq_queue_empty,
 };
 #endif

diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index a382953..63d2545 100644
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -60,6 +60,19 @@ struct cfq_meta_data {
  unsigned int siblings;

  struct cfq_driver_data cfq_driv_d;
+
+ /*
+  * rr list of queues with requests and the count of them
+  */
+ struct cfq_rb_root service_tree;
+ unsigned int busy_data;
+
+ struct cfq_data *active_data;
+
+ /*
+  * tunables,
+  */
+ unsigned int cfq_slice;
 };
 #endif

@@ -104,6 +117,18 @@ struct cfq_data {
  /* group_tree member for cfq_cgroup */
  struct rb_node group_node;

+ /* service_tree member */
+ struct rb_node rb_node;
+ /* service_tree key */
+ unsigned long rb_key;
+ /*
```

```
+  * slice parameter
+  */
+ unsigned long slice_end;
+ long slice_resid;
+
+ /* various state flags, see below */
+        unsigned int flags;
 #else
  struct cfq_driver_data cfq_driv_d;
 #endif
@@ -132,12 +157,20 @@ static inline struct request_queue * __cfq_container_of_queue(struct
work_struct
 typedef struct cfq_data *(cfq_init_queue_fn)(struct request_queue *, void *);
 typedef void (cfq_exit_queue_fn)(struct cfq_data *);
 typedef struct cfq_data *(cfq_search_data_fn)(void *, struct task_struct *);
+typedef int (cfq_dispatch_requests_fn)(struct cfq_data *, int);
+typedef int (cfq_completed_request_after_fn)(struct cfq_data *);
+typedef int (cfq_active_check_fn)(struct cfq_data *);
+typedef int (cfq_empty_fn)(struct cfq_data *);

 struct cfq_ops
 {
  cfq_init_queue_fn *cfq_init_queue_fn;
  cfq_exit_queue_fn *cfq_exit_queue_fn;
  cfq_search_data_fn *cfq_search_data_fn;
+ cfq_dispatch_requests_fn *cfq_dispatch_requests_fn;
+ cfq_completed_request_after_fn *cfq_completed_request_after_fn;
+ cfq_active_check_fn *cfq_active_check_fn;
+ cfq_empty_fn *cfq_empty_fn;
 };

 extern struct cfq_ops opt;
@@ -145,7 +178,13 @@ extern struct cfq_ops opt;

 extern struct cfq_data *__cfq_init_cfq_data(struct request_queue *q);
 extern void cfq_kick_queue(struct work_struct *work);
-extern void cfq_idle_slice_timer(unsigned long data);
 extern void __cfq_exit_data(struct cfq_data *cfqd);
+extern void cfq_rb_erase(struct rb_node *n, struct cfq_rb_root *root);
+extern void cfq_slice_expired(struct cfq_data *cfqd, int timed_out);
+extern int cfq_queue_dispatch_requests(struct cfq_data *cfqd, int force);
+extern int wait_request_checker(struct cfq_data *cfqd);
+extern int cfq_forced_dispatch(struct cfq_data *cfqd);
+extern int __cfq_idle_slice_timer(struct cfq_data *cfqd);
+extern void cfq_schedule_dispatch(struct cfq_data *cfqd);

 #endif  /* _LINUX_CFQ_IOSCHED_H */
```

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

Subject: [RFC][patch 11/11][CFQ-cgroup] entry/remove active cfq_data
Posted by Satoshi UCHIDA on Tue, 01 Apr 2008 09:42:35 GMT
View Forum Message <> Reply to Message

This patch controls whether cfq_data is active or not.
When cfq_data is not active and active cfq_queue is inserted into cfq_data,
cfq_data is activated.
When cfq_data is active and active cfq_queue is not exist,
cfq_data is deactivated.

The new cfq optional operations:
The "cfq_add_cfqq_opt_fn" defines a function that runs an additional process
when active queue is inserted into cfq_data.

The "cfq_del_cfqq_opt_fn" defines a function that runs an additional process
when active queue is removed in cfq_data.

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

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index f040c98..46f3635 100644
--- a/block/cfq-cgroup.c
+++ b/block/cfq-cgroup.c
@@ -741,6 +741,32 @@ static int cfq_cgroup_active_data_check(struct cfq_data *cfqd)
  return (cfqd->cfqmd->active_data == cfqd);
 }

+static void cfq_cgroup_add_cfqd_rr(struct cfq_data *cfqd)
+{
+ if (!cfq_cfqd_on_rr(cfqd)) {
+  cfq_mark_cfqd_on_rr(cfqd);
+  cfqd->cfqmd->busy_data++;
+
+  cfq_cgroup_service_tree_add(cfqd, 0);
+ }
+}
+
+
+static void cfq_cgroup_del_cfqd_rr(struct cfq_data *cfqd)
+{
+ if (RB_EMPTY_ROOT(&cfqd->service_tree.rb)) {
```

```
+  struct cfq_meta_data *cfqdd = cfqd->cfqmd;
+  BUG_ON(!cfq_cfqd_on_rr(cfqd));
+  cfq_clear_cfqd_on_rr(cfqd);
+  if (!RB_EMPTY_NODE(&cfqd->rb_node)) {
+   cfq_rb_erase(&cfqd->rb_node,
+         &cfqdd->service_tree);
+  }
+  BUG_ON(!cfqdd->busy_data);
+  cfqdd->busy_data--;
+ }
+}
+
 struct cfq_ops opt = {
  .cfq_init_queue_fn = __cfq_cgroup_init_queue,
  .cfq_exit_queue_fn = __cfq_cgroup_exit_data,
@@ -749,4 +775,6 @@ struct cfq_ops opt = {
  .cfq_completed_request_after_fn = cfq_cgroup_completed_request_after,
  .cfq_empty_fn = cfq_cgroup_queue_empty,
  .cfq_active_check_fn = cfq_cgroup_active_data_check,
+ .cfq_add_cfqq_opt_fn =  cfq_cgroup_add_cfqd_rr,
+ .cfq_del_cfqq_opt_fn =  cfq_cgroup_del_cfqd_rr,
 };
diff --git a/block/cfq-iosched.c b/block/cfq-iosched.c
index 505e425..8f5227f 100644
--- a/block/cfq-iosched.c
+++ b/block/cfq-iosched.c
@@ -492,6 +492,9 @@ static void cfq_add_cfqq_rr(struct cfq_data *cfqd, struct cfq_queue *cfqq)
  cfqd->busy_queues++;

  cfq_resort_rr_list(cfqd, cfqq);
+
+ if (opt.cfq_add_cfqq_opt_fn)
+  opt.cfq_add_cfqq_opt_fn(cfqd);
 }

 /*
@@ -508,6 +511,9 @@ static void cfq_del_cfqq_rr(struct cfq_data *cfqd, struct cfq_queue *cfqq)

  BUG_ON(!cfqd->busy_queues);
  cfqd->busy_queues--;
+
+ if (opt.cfq_del_cfqq_opt_fn)
+  opt.cfq_del_cfqq_opt_fn(cfqd);
 }

 /*
diff --git a/include/linux/cfq-iosched.h b/include/linux/cfq-iosched.h
index 63d2545..ed35050 100644
```

```
--- a/include/linux/cfq-iosched.h
+++ b/include/linux/cfq-iosched.h
@@ -161,6 +161,8 @@ typedef int (cfq_dispatch_requests_fn)(struct cfq_data *, int);
 typedef int (cfq_completed_request_after_fn)(struct cfq_data *);
 typedef int (cfq_active_check_fn)(struct cfq_data *);
 typedef int (cfq_empty_fn)(struct cfq_data *);
+typedef void (cfq_add_cfqq_opt_fn)(struct cfq_data *);
+typedef void (cfq_del_cfqq_opt_fn)(struct cfq_data *);

 struct cfq_ops
 {
@@ -171,6 +173,8 @@ struct cfq_ops
   cfq_completed_request_after_fn *cfq_completed_request_after_fn;
   cfq_active_check_fn *cfq_active_check_fn;
   cfq_empty_fn *cfq_empty_fn;
+ cfq_add_cfqq_opt_fn *cfq_add_cfqq_opt_fn;
+ cfq_del_cfqq_opt_fn *cfq_del_cfqq_opt_fn;
 };

 extern struct cfq_ops opt;
```

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

---

Subject: Re: [RFC][patch 3/11][CFQ-cgroup] Introduce cgroup subsystem
Posted by Paul Menage on Wed, 02 Apr 2008 22:41:58 GMT
View Forum Message <> Reply to Message

On Tue, Apr 1, 2008 at 2:32 AM, Satoshi UCHIDA <s-uchida@ap.jp.nec.com> wrote:
> This patch introduces a simple cgroup subsystem.
> New cgroup subsystem is called cfq_cgroup.
>
>   Signed-off-by: Satoshi UCHIDA <uchida@ap.jp.nec.com>
>
> diff --git a/block/Makefile b/block/Makefile
> index 5a43c7d..ea07b46 100644
> --- a/block/Makefile
> +++ b/block/Makefile
> @@ -11,6 +11,7 @@ obj-$(CONFIG_IOSCHED_NOOP)   += noop-iosched.o
>   obj-$(CONFIG_IOSCHED_AS)   += as-iosched.o
>   obj-$(CONFIG_IOSCHED_DEADLINE) += deadline-iosched.o
>   obj-$(CONFIG_IOSCHED_CFQ)   += cfq-iosched.o
> +obj-$(CONFIG_CGROUP_CFQ)   += cfq-cgroup.o
>
>   obj-$(CONFIG_BLK_DEV_IO_TRACE) += blktrace.o

```
>   obj-$(CONFIG_BLOCK_COMPAT)     += compat_ioctl.o
>  diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
>  new file mode 100644
>  index 0000000..cea2b92
>  --- /dev/null
>  +++ b/block/cfq-cgroup.c
>  @@ -0,0 +1,57 @@
>  +/*
>  + *  CFQ CGROUP disk scheduler.
>  + *
>  + *     This program is a wrapper program that is
>  + *     extend CFQ disk scheduler for handling
>  + *     cgroup subsystem.
>  + *
>  + *     This program is based on original CFQ code.
>  + *
>  + *  Copyright (C) 2008 Satoshi UCHIDA <s-uchida@ap.jp.nec.com>
>  + *   and NEC Corp.
>  + */
>  +
>  +#include <linux/blkdev.h>
>  +#include <linux/cgroup.h>
>  +#include <linux/cfq-iosched.h>
>  +
>  +struct cfq_cgroup {
>  +       struct cgroup_subsys_state css;
>  +};
>  +
>  +
>  +static inline struct cfq_cgroup *cgroup_to_cfq_cgroup(struct cgroup *cont)
>  +{
>  +       return container_of(cgroup_subsys_state(cont, cfq_cgroup_subsys_id),
>  +                           struct cfq_cgroup, css);
>  +}
>  +
>  +static struct cgroup_subsys_state *
>  +cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
>  +{
>  +       struct cfq_cgroup *cfqc;
>  +
>  +       if (!capable(CAP_SYS_ADMIN))
>  +               return ERR_PTR(-EPERM);
>  +
>  +       if (!cgroup_is_descendant(cont))
>  +               return ERR_PTR(-EPERM);
```

What are these checks for? Cgroups already provides filesystem permissions to control directory creation, and the "descendant" check

looks like it may have been cut/pasted from the nsproxy subsystem.

```
> +
> +       cfqc = kzalloc(sizeof(struct cfq_cgroup), GFP_KERNEL);
> +       if (unlikely(!cfqc))
> +               return ERR_PTR(-ENOMEM);
> +
> +       return &cfqc->css;
> +}
> +
> +static void cfq_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> +       kfree(cgroup_to_cfq_cgroup(cont));
> +}
> +
> +struct cgroup_subsys cfq_cgroup_subsys = {
> +       .name = "cfq_cgroup",
> +       .create = cfq_cgroup_create,
> +       .destroy = cfq_cgroup_destroy,
> +       .subsys_id = cfq_cgroup_subsys_id,
> +};
> diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> index 1ddebfc..5d2e991 100644
> --- a/include/linux/cgroup_subsys.h
> +++ b/include/linux/cgroup_subsys.h
> @@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
>  #endif
>
>  /* */
> +
> +#ifdef CONFIG_CGROUP_CFQ
> +SUBSYS(cfq_cgroup)
> +#endif
> +
> +/* */
```

To fit with the convention for other subsystems, simply "cfq" would be a better name than "cfq_cgroup". (Clearly it's a cgroup subsystem from context).

Is this subsystem meant to allow you to control any device that uses CFQ, or is it specific to disks? It would be nice to be able to allow different groups have different guarantees on different disks.

Paul

_____
Containers mailing list

Subject: Re: [RFC][patch 3/11][CFQ-cgroup] Introduce cgroup subsystem
Posted by Li Zefan on Thu, 03 Apr 2008 02:39:09 GMT
View Forum Message <> Reply to Message

Satoshi UCHIDA wrote:
> Thank you for reply.
>
>>> +
>>> +static struct cgroup_subsys_state *
>>> +cfq_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
>>> +{
>>> + struct cfq_cgroup *cfqc;
>>> +
>>> + if (!capable(CAP_SYS_ADMIN))
>>> + return ERR_PTR(-EPERM);
>>> +
>>> + if (!cgroup_is_descendant(cont))
>>> + return ERR_PTR(-EPERM);
>> What are these checks for? Cgroups already provides filesystem
>> permissions to control directory creation, and the "descendant" check
>> looks like it may have been cut/pasted from the nsproxy subsystem.
>>
>
> This code was referred one of io-throttle.
> Is it not necessary these checks?
> IF not necessary, remove this code.
>
>>> /* */
>>> +
>>> +#ifdef CONFIG_CGROUP_CFQ
>>> +SUBSYS(cfq_cgroup)
>>> +#endif
>>> +
>>> +/* */
>> To fit with the convention for other subsystems, simply "cfq" would be
>> a better name than "cfq_cgroup". (Clearly it's a cgroup subsystem from
>> context).
>>
>
> Ok, I change name.
> I hesitated whether using "_cgroup".
> The cpuset and the cpuacct does not use it,
> but cpu and memory uses it(cpu_cgroup and mem_cgroup).
> In this patchset, I select the latter case.

>

+struct cgroup_subsys cfq_cgroup_subsys = {
+ .name = "cfq_cgroup",
+ ...
+};

but memory controller has the name 'memory', similar for cgroup sched.

So we do this:
 mount -t cgroup -omemory xxx /dev/memcg
but not:
 mount -t cgroup -omemory_cgroup xxx /dev/memcg

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

---

Subject: Re: [RFC][patch 3/11][CFQ-cgroup] Introduce cgroup subsystem
Posted by Paul Menage on Thu, 03 Apr 2008 15:31:25 GMT
View Forum Message <> Reply to Message

On Wed, Apr 2, 2008 at 7:31 PM, Satoshi UCHIDA <s-uchida@ap.jp.nec.com> wrote:
> >
> > What are these checks for? Cgroups already provides filesystem
> > permissions to control directory creation, and the "descendant" check
> > looks like it may have been cut/pasted from the nsproxy subsystem.
> >
>
> This code was referred one of io-throttle.
> Is it not necessary these checks?
> IF not necessary, remove this code.
>

Yes, I think these checks should be removed. Especially the descendant check.

Paul

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

---

Subject: Re: [RFC][patch 8/11][CFQ-cgroup] Control cfq_data per cgroup
Posted by Paul Menage on Thu, 03 Apr 2008 15:35:04 GMT
View Forum Message <> Reply to Message

On Tue, Apr 1, 2008 at 2:38 AM, Satoshi UCHIDA <s-uchida@ap.jp.nec.com> wrote:
> +
> +static void *cfq_cgroup_init_cfq_data(struct cfq_cgroup *cfqc, struct cfq_data *cfqd)
> +{
> +	struct cgroup *child;
> +
> +	/* setting cfq_data for cfq_cgroup */
> +	if (!cfqc) {
> +		cfqc = cgroup_to_cfq_cgroup(get_root_subsys(&cfq_cgroup_subsys));

Rather than adding get_root_subsys(), can't you just keep a reference
locally to your root subsystem state?

Paul

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

---

Subject: RE: [RFC][patch 8/11][CFQ-cgroup] Control cfq_data per cgroup
Posted by Satoshi UCHIDA on Fri, 04 Apr 2008 06:20:39 GMT
View Forum Message <> Reply to Message

\>
> On Tue, Apr 1, 2008 at 2:38 AM, Satoshi UCHIDA <s-uchida@ap.jp.nec.com>
> wrote:
> > > +
> > > +static void *cfq_cgroup_init_cfq_data(struct cfq_cgroup *cfqc, struct
> cfq_data *cfqd)
> > > +{
> > > +	struct cgroup *child;
> > > +
> > > +	/* setting cfq_data for cfq_cgroup */
> > > +	if (!cfqc) {
> > > +		cfqc =
> cgroup_to_cfq_cgroup(get_root_subsys(&cfq_cgroup_subsys));
>
> Rather than adding get_root_subsys(), can't you just keep a reference
> locally to your root subsystem state?
>

If the cfqc has not specific address, namely cfqc is null,
the cfq_cgroup_init_cfq_data function is called when new device is plugged.
In this time, it needs to relate new cfq_data data with top cfq_cgroup data.
Probably, a running program will be "insmod" in its time.
However, its program is not in root group.

On the supposition that only current interface is usedm,
If using current process, the top cgroup can be calculated by

  task_cgroup(current, cfq_subsys_id)->top_cgroup .

Therefore cfq_cgroup data of top cgroup is calculated by

  cgroup_to_cfq_cgroup(task_cgroup(current, cfq_subsys_id)->top_cgroup) .

However, It would be bad to use "current" variable in order to
calculate cfq_cgroup data of top cgroup.
Because relationship between "calculating cfq_cgroup data of top cgroup" and
"running(current) task" is weak.

So that you say, root subsystem state maybe keep a reference locally.
For example, create a variable for root subsystem state and
store the pointer when making subsystem state first.

However, I think that it is smart to calculate root group of subsystems
when needing its information.
Does the current code have any problem?

Satoshi UCHIDA.

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

---

Subject: Re: [RFC][patch 8/11][CFQ-cgroup] Control cfq_data per cgroup
Posted by Paul Menage on Fri, 04 Apr 2008 09:00:10 GMT
View Forum Message <> Reply to Message

On Thu, Apr 3, 2008 at 11:20 PM, Satoshi UCHIDA <s-uchida@ap.jp.nec.com> wrote:
>
> So that you say, root subsystem state maybe keep a reference locally.
> For example, create a variable for root subsystem state and
> store the pointer when making subsystem state first.

Yes, that's what I'm suggesting.

>
> However, I think that it is smart to calculate root group of subsystems

> when needing its information.

I don't see why it's better to go through the cgroup subsystem to
retrieve the reference.

Paul

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

---

Subject: RE: [RFC][patch 8/11][CFQ-cgroup] Control cfq_data per cgroup
Posted by Satoshi UCHIDA on Fri, 04 Apr 2008 09:46:53 GMT
View Forum Message <> Reply to Message

Hi,

>
> On Thu, Apr 3, 2008 at 11:20 PM, Satoshi UCHIDA <s-uchida@ap.jp.nec.com>
> wrote:
> >
> > So that you say, root subsystem state maybe keep a reference locally.
> > For example, create a variable for root subsystem state and
> > store the pointer when making subsystem state first.
>
> Yes, that's what I'm suggesting.
>
> >
> > However, I think that it is smart to calculate root group of subsystems
> > when needing its information.
>
> I don't see why it's better to go through the cgroup subsystem to
> retrieve the reference.
>

If program have temporary data, it may cause inconsistency.
I think that information should be referred to original(control mechanism).

Information of cgroup is controlled at cgroup subsystem (and cgroupfs_root).
So, I think that it's better to query into cgroup subsystem.

Now, cgroup susbsystem would not change root group.
Therefore, system would also runs by your suggestion.

I will consider this issue.

Thanks,

Satoshi UCHIDA.

_____

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