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 */
```