Subject: [RFC][patch 11/12][CFQ-cgroup] Control service tree: Main functions
Posted by Satoshi UCHIDA on Thu, 03 Apr 2008 07:17:31 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>

---
block/cfq-cgroup.c | 326 ++-
block/cfq-iosched.c | 89 ++++++++++---
include/linux/cfq-iosched.h | 41 ++++++-
3 files changed, 434 insertions(+), 22 deletions(-)

```
diff --git a/block/cfq-cgroup.c b/block/cfq-cgroup.c
index 6a8a219..27a9a7a 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 */
--
1.5.4.1
```

_____

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