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