
Subject: [PATCH] cfq: get rid of cfqq hash
Posted by [Vasily Tarasov](#) on Tue, 24 Apr 2007 13:53:11 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Vasily Tarasov <vtaras@openvz.org>

cfq hash is no more necessary. We always can get cfqq from io context. cfq_get_io_context_noalloc() function is introduced, because we don't want to allocate cic on merging and checking may_queue. In order to identify sync queue we've used hash key = CFQ_KEY_ASYNC. Since hash is eliminated we need to use other criterion: sync flag for queue is added. In all places where we dig in rb_tree we're in current context, so no additional locking is required.

Advantages of this patch: no additional memory for hash, no seeking in hash, code is cleaner. But it is necessary now to seek cic in per-ioc rbtree, but it is faster:

- most processes work only with few devices
- most systems have only few block devices
- it is a rb-tree

Signed-off-by: Vasily Tarasov <vtaras@openvz.org>

block/cfq-iosched.c | 152 ++++++-----
1 files changed, 52 insertions(+), 100 deletions(-)

--- linux-2.6.21-rc7-git6/block/cfq-iosched.c.hashrm 2007-04-24 16:40:45.000000000 +0400

+++ linux-2.6.21-rc7-git6/block/cfq-iosched.c 2007-04-24 17:17:41.000000000 +0400

@@ -9,7 +9,6 @@

```
#include <linux/module.h>  
#include <linux/blkdev.h>  
#include <linux/elevator.h>  
-#include <linux/hash.h>  
#include <linux/rbtree.h>  
#include <linux/ioprio.h>
```

@@ -29,15 +28,6 @@ static int cfq_slice_idle = HZ / 125;

```
#define CFQ_IDLE_GRACE (HZ / 10)
```

```
#define CFQ_SLICE_SCALE (5)
```

```
+#define CFQ_KEY_ASYNC (0)
```

```
-  
-/*
```

```
- * for the hash of cfqq inside the cfqd
```

```
- */
```

```
+#define CFQ_QHASH_SHIFT 6
```

```

-#define CFQ_QHASH_ENTRIES (1 << CFQ_QHASH_SHIFT)
-#define list_entry_qhash(entry) hlist_entry((entry), struct cfq_queue, cfq_hash)
-
#define list_entry_cfqq(ptr) list_entry((ptr), struct cfq_queue, cfq_list)

#define RQ_CIC(rq) ((struct cfq_io_context*)(rq)->elevator_private)
@@ -59,11 +49,6 @@ static struct completion *ioc_gone;
#define cfq_cfqq_dispatched(cfqq) \
((cfqq)->on_dispatch[ASYNC] + (cfqq)->on_dispatch[SYNC])

-#define cfq_cfqq_class_sync(cfqq) ((cfqq)->key != CFQ_KEY_ASYNC)
-
-#define cfq_cfqq_sync(cfqq) \
- (cfq_cfqq_class_sync(cfqq) || (cfqq)->on_dispatch[SYNC])
-
#define sample_valid(samples) ((samples) > 80)

/*
@@ -81,11 +66,6 @@ struct cfq_data {
    struct list_head idle_rr;
    unsigned int busy_queues;

- /*
-  * cfqq lookup hash
-  */
- struct hlist_head *cfq_hash;
-
    int rq_in_driver;
    int hw_tag;

@@ -127,10 +107,6 @@ struct cfq_queue {
    atomic_t ref;
    /* parent cfq_data */
    struct cfq_data *cfqd;
- /* cfqq lookup hash */
- struct hlist_node cfq_hash;
- /* hash key */
- unsigned int key;
    /* member of the rr/busy/cur/idle cfqd list */
    struct list_head cfq_list;
    /* sorted list of pending requests */
@@ -172,6 +148,7 @@ enum cfqq_state_flags {
    CFQ_CFQQ_FLAG_prio_changed, /* task priority has changed */
    CFQ_CFQQ_FLAG_queue_new, /* queue never been serviced */
    CFQ_CFQQ_FLAG_slice_new, /* no requests dispatched in slice */
+ CFQ_CFQQ_FLAG_sync, /* synchronous queue */
};

```

```

#define CFQ_CFQQ_FNS(name) \
@@ -198,11 +175,14 @@ CFQ_CFQQ_FNS(idle_window);
CFQ_CFQQ_FNS(prio_changed);
CFQ_CFQQ_FNS(queue_new);
CFQ_CFQQ_FNS(slice_new);
+CFQ_CFQQ_FNS(sync);
#undef CFQ_CFQQ_FNS

-static struct cfq_queue *cfq_find_cfq_hash(struct cfq_data *, unsigned int, unsigned short);
+static struct cfq_io_context *
+cfq_get_io_context_noalloc(struct cfq_data *, struct task_struct *);
static void cfq_dispatch_insert(request_queue_t *, struct request *);
-static struct cfq_queue *cfq_get_queue(struct cfq_data *cfqd, unsigned int key, struct task_struct
*tsk, gfp_t gfp_mask);
+static struct cfq_queue *cfq_get_queue(struct cfq_data *cfqd,
+ int is_sync, struct task_struct *tsk, gfp_t gfp_mask);

/*
 * scheduler run of queue, if there are requests pending and no one in the
@@ -221,17 +201,6 @@ static int cfq_queue_empty(request_queue
return !cfqd->busy_queues;
}

-static inline pid_t cfq_queue_pid(struct task_struct *task, int rw, int is_sync)
-{
- /*
- * Use the per-process queue, for read requests and synchronous writes
- */
- if (!(rw & REQ_RW) || is_sync)
- return task->pid;
-
- return CFQ_KEY_ASYNC;
-}
-
/*
 * 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
@@ -445,7 +414,7 @@ static void cfq_resort_rr_list(struct cf
n = n->next;
}
list_add_tail(&cfqq->cfq_list, n);
- } else if (!cfq_cfqq_class_sync(cfqq)) {
+ } else if (!cfq_cfqq_sync(cfqq)) {
/*
 * async queue always goes to the end. this wont be overly
 * unfair to writes, as the sort of the sync queue wont be
@@ -463,7 +432,7 @@ static void cfq_resort_rr_list(struct cf
while ((n = n->prev) != list) {

```

```

struct cfq_queue *__cfqq = list_entry_cfqq(n);

- if (!cfq_cfqq_class_sync(cfqq) || !__cfqq->service_last)
+ if (!cfq_cfqq_sync(cfqq) || !__cfqq->service_last)
    break;
    if (time_before(__cfqq->service_last, cfqq->service_last))
        break;
@@ -546,10 +515,14 @@ static struct request *
cfq_find_rq_fmerge(struct cfq_data *cfqd, struct bio *bio)
{
    struct task_struct *tsk = current;
- pid_t key = cfq_queue_pid(tsk, bio_data_dir(bio), bio_sync(bio));
+ struct cfq_io_context *cic;
    struct cfq_queue *cfqq;

- cfqq = cfq_find_cfq_hash(cfqd, key, tsk->ioprio);
+ cic = cfq_get_io_context_noalloc(cfqd, tsk);
+ if (!cic)
+ return NULL;
+
+ cfqq = cic->cfqq[bio_sync(bio)];
    if (cfqq) {
        sector_t sector = bio->bi_sector + bio_sectors(bio);

@@ -642,9 +615,8 @@ static int cfq_allow_merge(request_queue
    struct bio *bio)
{
    struct cfq_data *cfqd = q->elevator->elevator_data;
- const int rw = bio_data_dir(bio);
+ struct cfq_io_context *cic;
    struct cfq_queue *cfqq;
- pid_t key;

/*
 * Disallow merge of a sync bio into an async request.
@@ -656,9 +628,11 @@ static int cfq_allow_merge(request_queue
 * Lookup the cfqq that this bio will be queued with. Allow
 * merge only if rq is queued there.
 */
- key = cfq_queue_pid(current, rw, bio_sync(bio));
- cfqq = cfq_find_cfq_hash(cfqd, key, current->ioprio);
+ cic = cfq_get_io_context_noalloc(cfqd, current);
+ if (!cic)
+ return 0;

+ cfqq = cic->cfqq[bio_sync(bio)];
    if (cfqq == RQ_CFQQ(rq))
        return 1;

```



```

-}
-
-static struct cfq_queue *
-cfq_find_cfq_hash(struct cfq_data *cfqd, unsigned int key, unsigned short prio)
-{
- return __cfq_find_cfq_hash(cfqd, key, prio, hash_long(key, CFQ_QHASH_SHIFT));
-}
-
static void cfq_free_io_context(struct io_context *ioc)
{
struct cfq_io_context * __cic;
@@ -1297,7 +1246,7 @@ static inline void changed_ioprio(struct
cfqq = cic->cfqq[ASYNC];
if (cfqq) {
struct cfq_queue *new_cfqq;
- new_cfqq = cfq_get_queue(cfqd, CFQ_KEY_ASYNC, cic->ioc->task,
+ new_cfqq = cfq_get_queue(cfqd, ASYNC, cic->ioc->task,
GFP_ATOMIC);
if (new_cfqq) {
cic->cfqq[ASYNC] = new_cfqq;
@@ -1329,16 +1278,16 @@ static void cfq_ioc_set_ioprio(struct io
}

static struct cfq_queue *
-cfq_get_queue(struct cfq_data *cfqd, unsigned int key, struct task_struct *tsk,
+cfq_get_queue(struct cfq_data *cfqd, int is_sync, struct task_struct *tsk,
gfp_t gfp_mask)
{
- const int hashval = hash_long(key, CFQ_QHASH_SHIFT);
struct cfq_queue *cfqq, *new_cfqq = NULL;
- unsigned short ioprio;
+ struct cfq_io_context *cic;

retry:
- ioprio = tsk->ioprio;
- cfqq = __cfq_find_cfq_hash(cfqd, key, ioprio, hashval);
+ cic = cfq_get_io_context_noalloc(cfqd, tsk);
+ /* cic always exists here */
+ cfqq = cic->cfqq[is_sync];

if (!cfqq) {
if (new_cfqq) {
@@ -1363,20 +1312,19 @@ retry:

memset(cfqq, 0, sizeof(*cfqq));

- INIT_HLIST_NODE(&cfqq->cfq_hash);
INIT_LIST_HEAD(&cfqq->cfq_list);

```

```
INIT_LIST_HEAD(&cfqq->fifo);
```

```
- cfqq->key = key;  
- hlist_add_head(&cfqq->cfq_hash, &cfqd->cfq_hash[hashval]);  
  atomic_set(&cfqq->ref, 0);  
  cfqq->cfqd = cfqd;
```

```
- if (key != CFQ_KEY_ASYNC)  
+ if (is_sync)  
  cfq_mark_cfqq_idle_window(cfqq);  
  
  cfq_mark_cfqq_prio_changed(cfqq);  
  cfq_mark_cfqq_queue_new(cfqq);  
+ if (is_sync)  
+ cfq_mark_cfqq_sync(cfqq);  
  cfq_init_prio_data(cfqq);  
}
```

```
@@ -1506,6 +1454,19 @@ err:  
  return NULL;  
}
```

```
+static struct cfq_io_context *  
+cfq_get_io_context_noalloc(struct cfq_data *cfqd, struct task_struct *tsk)  
+{  
+ struct cfq_io_context *cic = NULL;  
+ struct io_context *ioc;  
+  
+ ioc = tsk->io_context;  
+ if (ioc)  
+ cic = cfq_cic_rb_lookup(cfqd, ioc);  
+  
+ return cic;  
+}  
+  
static void  
cfq_update_io_thinktime(struct cfq_data *cfqd, struct cfq_io_context *cic)  
{  
@@ -1795,10 +1756,8 @@ static int cfq_may_queue(request_queue_t  
{  
  struct cfq_data *cfqd = q->elevator->elevator_data;  
  struct task_struct *tsk = current;  
+ struct cfq_io_context *cic;  
  struct cfq_queue *cfqq;  
- unsigned int key;  
-  
- key = cfq_queue_pid(tsk, rw, rw & REQ_RW_SYNC);
```

```

/*
 * don't force setup of a queue from here, as a call to may_queue
@@ -1806,7 +1765,11 @@ static int cfq_may_queue(request_queue_t
 * so just lookup a possibly existing queue, or return 'may queue'
 * if that fails
 */
- cfqq = cfq_find_cfq_hash(cfqd, key, tsk->ioprio);
+ cic = cfq_get_io_context_noalloc(cfqd, tsk);
+ if (!cic)
+ return ELV_MQUEUE_MAY;
+
+ cfqq = cic->cfqq[rw & REQ_RW_SYNC];
  if (cfqq) {
    cfq_init_prio_data(cfqq);
    cfq_prio_boost(cfqq);
@@ -1850,7 +1813,6 @@ cfq_set_request(request_queue_t *q, stru
 struct cfq_io_context *cic;
 const int rw = rq_data_dir(rq);
 const int is_sync = rq_is_sync(rq);
- pid_t key = cfq_queue_pid(tsk, rw, is_sync);
 struct cfq_queue *cfqq;
 unsigned long flags;

@@ -1864,7 +1826,8 @@ cfq_set_request(request_queue_t *q, stru
 goto queue_fail;

  if (!cic->cfqq[is_sync]) {
- cfqq = cfq_get_queue(cfqd, key, tsk, gfp_mask);
+ cfqq = cfq_get_queue(cfqd, is_sync, tsk, gfp_mask);
+
  if (!cfqq)
    goto queue_fail;

@@ -2000,7 +1963,6 @@ static void cfq_exit_queue(elevator_t *e

  cfq_shutdown_timer_wq(cfqd);

- kfree(cfqd->cfq_hash);
  kfree(cfqd);
}

@@ -2023,13 +1985,6 @@ static void *cfq_init_queue(request_queu
 INIT_LIST_HEAD(&cfqd->idle_rr);
 INIT_LIST_HEAD(&cfqd->cic_list);

- cfqd->cfq_hash = kmalloc_node(sizeof(struct hlist_head) * CFQ_QHASH_ENTRIES,
GFP_KERNEL, q->node);
- if (!cfqd->cfq_hash)

```



```
- goto out_free;
-
- for (i = 0; i < CFQ_QHASH_ENTRIES; i++)
- INIT_HLIST_HEAD(&cfqd->cfq_hash[i]);
-
  cfqd->queue = q;

  init_timer(&cfqd->idle_slice_timer);
@@ -2053,9 +2008,6 @@ static void *cfq_init_queue(request_queu
  cfqd->cfq_slice_idle = cfq_slice_idle;

  return cfqd;
-out_free:
- kfree(cfqd);
- return NULL;
}

static void cfq_slab_kill(void)
```
