
Subject: Re: [PATCH] cfq: get rid of cfqq hash
Posted by [Jens Axboe](#) on Tue, 24 Apr 2007 17:22:29 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, Apr 24 2007, Jens Axboe wrote:

> On Tue, Apr 24 2007, Vasily Tarasov wrote:

> > 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

>

> Vasily, this is still not against the CFQ branch, I get tons of rejects:

>

> axboe@nelson:/src/linux-2.6-block \$ patch -p1 --dry-run < ~/foo

> [...]

> 10 out of 27 hunks FAILED -- saving rejects to file

> block/cfq-iosched.c.rej

>

> If you don't want to use the git tree, then just grab

>

> <http://brick.kernel.dk/snaps/cfq-update-20070424>

>

> and apply it to 2.6.21-rc7-gitX (latest) and provide a diff against

> that. Thanks!

I merged it myself, care to double check it? I'll do some testing on it tomorrow, and integrate if I'm happy with it.

Have you done any multi disk testing? scsi_debug can be quite handy for such things, testing thousands of io_contexts and disks. Just be sure to use delay=1 and fake_rw=1.

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

index 8093733..e42c09b 100644

--- a/block/cfq-iosched.c

+++ b/block/cfq-iosched.c

```

@@ -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>

@@ -38,14 +37,6 @@ static int cfq_slice_idle = HZ / 125;

#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 RQ_CIC(rq) ((struct cfq_io_context*)(rq)->elevator_private)
#define RQ_CFQQ(rq) ((rq)->elevator_private2)

@@ -62,8 +53,6 @@ static struct completion *ioc_gone;
#define ASYNC (0)
#define SYNC (1)

-#define cfq_cfqq_sync(cfqq) ((cfqq)->key != CFQ_KEY_ASYNC)
-
#define sample_valid(samples) ((samples) > 80)

/*
@@ -90,11 +79,6 @@ struct cfq_data {
    struct cfq_rb_root service_tree;
    unsigned int busy_queues;

- /*
-  * cfqq lookup hash
-  */
- struct hlist_head *cfq_hash;
-
    int rq_in_driver;
    int sync_flight;
    int hw_tag;
@@ -138,10 +122,6 @@ 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;
  /* service_tree member */
  struct rb_node rb_node;
  /* service_tree key */
@@ -186,6 +166,7 @@ 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) \
@@ -212,11 +193,14 @@ 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 *, unsigned int, struct task_struct *, gfp_t);
+static struct cfq_queue *cfq_get_queue(struct cfq_data *, int,
+ struct task_struct *, gfp_t);

/*
 * scheduler run of queue, if there are requests pending and no one in the
@@ -235,17 +219,6 @@ static int cfq_queue_empty(request_queue_t *q)
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
@@ -602,10 +575,14 @@ 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);

@@ -699,9 +676,8 @@ static int cfq_allow_merge(request_queue_t *q, struct request *rq,
    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.
@@ -713,9 +689,11 @@ static int cfq_allow_merge(request_queue_t *q, struct request *rq,
     * 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;

@@ -1148,37 +1126,9 @@ static void cfq_put_queue(struct cfq_queue *cfqq)
    cfq_schedule_dispatch(cfqd);
}

```

```

- /*
-  * it's on the empty list and still hashed
-  */
- hlist_del(&cfqq->cfq_hash);
- kmem_cache_free(cfq_pool, cfqq);
- }

-static struct cfq_queue *
- __cfq_find_cfq_hash(struct cfq_data *cfqd, unsigned int key, unsigned int prio,
-   const int hashval)
- {
-   struct hlist_head *hash_list = &cfqd->cfq_hash[hashval];
-   struct hlist_node *entry;
-   struct cfq_queue * __cfqq;
-   -
-   hlist_for_each_entry(__cfqq, entry, hash_list, cfq_hash) {
-   const unsigned short __p = IOPRIO_PRIO_VALUE(__cfqq->org_ioprio_class,
-   __cfqq->org_ioprio);
-   -
-   if (__cfqq->key == key && (__p == prio || !prio))
-   return __cfqq;
-   }
-   -
-   return NULL;
- }
-
-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;
@@ -1336,7 +1286,7 @@ static inline void changed_ioprio(struct cfq_io_context *cic)
    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;
@@ -1368,16 +1318,16 @@ static void cfq_ioc_set_ioprio(struct io_context *ioc)
    }

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) {
@@ -1402,17 +1352,16 @@ retry:

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

- INIT_HLIST_NODE(&cfqq->cfq_hash);
- RB_CLEAR_NODE(&cfqq->rb_node);
- 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);
+ else
+   cfq_mark_cfqq_sync(cfqq);

    cfq_mark_cfqq_prio_changed(cfqq);
    cfq_mark_cfqq_queue_new(cfqq);
@@ -1548,6 +1497,19 @@ 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)
{
@@ -1839,10 +1801,8 @@ static int cfq_may_queue(request_queue_t *q, int rw)
{
    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
@@ -1850,7 +1810,11 @@ static int cfq_may_queue(request_queue_t *q, int rw)
 * 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);
@@ -1894,7 +1858,6 @@ cfq_set_request(request_queue_t *q, struct request *rq, gfp_t
gfp_mask)
    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;

@@ -1908,7 +1871,8 @@ cfq_set_request(request_queue_t *q, struct request *rq, gfp_t
gfp_mask)
    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;

@@ -2044,14 +2008,12 @@ static void cfq_exit_queue(elevator_t *e)

    cfq_shutdown_timer_wq(cfqd);

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

static void *cfq_init_queue(request_queue_t *q)
{
    struct cfq_data *cfqd;
- int i;

    cfqd = kmalloc_node(sizeof(*cfqd), GFP_KERNEL, q->node);
    if (!cfqd)
@@ -2062,13 +2024,6 @@ static void *cfq_init_queue(request_queue_t *q)
    cfqd->service_tree = CFQ_RB_ROOT;
    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);
@@ -2092,9 +2047,6 @@ static void *cfq_init_queue(request_queue_t *q)
    cfqd->cfq_slice_idle = cfq_slice_idle;

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

static void cfq_slab_kill(void)

```


--

Jens Axboe
