Subject: [RFC][PATCH 0/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:53:34 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

The following patchset introduces I/O bandwidth controlling subsystem
for the CGroups framework based on the CFQ scheduler.
User can assign a priority from 0 to 7 to a cgroup, and I/O bandwidth
will be adjusted proportionally.
This effect is achieved by introducing a certain modifications to CFQ.
Now there is not only per process time slice, but also per cgroup
time slice. During cgroup's time slice only processes from current
cgroup can add requests to the queue. Inside cgroups's time slice
processes are managed based on (per-process) time slices (as usual in CFQ).
This is why we call this approach "two level CFQ".

Such kind of system works in OpenVZ project.

Patch is against 2.6.25-rc2-mm1

Comments, suggestions, criticism are all welcome.

Subject: [RFC][PATCH 1/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:41 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Simply moves CFQ data structure definitions to the header file.

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

```
--- /dev/null 2008-02-14 08:57:49.255923728 -0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h 2008-02-15 01:03:38.000000000 -0500
@@ -0,0 +1,103 @@
+#ifndef _LINUX_CFQ_IOSCHED_H
+#define _LINUX_CFQ_IOSCHED_H
+
+/*
+ * Most of our rbtree usage is for sorting with min extraction, so
+ * if we cache the leftmost node we don't have to walk down the tree
+ * to find it. Idea borrowed from Ingo Molnars CFS scheduler. We should
```

```
+ * move this into the elevator for the rq sorting as well.
+ */
+struct cfq_rb_root {
+ struct rb_root rb;
+ struct rb_node *left;
+};
+#define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }
+
+/*
+ * Per block device queue structure
+ */
+struct cfq_data {
+ struct request_queue *queue;
+
+ /*
+  * rr list of queues with requests and the count of them
+  */
+ struct cfq_rb_root service_tree;
+ unsigned int busy_queues;
+
+ int rq_in_driver;
+ int sync_flight;
+ int hw_tag;
+
+ /*
+  * idle window management
+  */
+ struct timer_list idle_slice_timer;
+ struct work_struct unplug_work;
+
+ struct cfq_queue *active_queue;
+ struct cfq_io_context *active_cic;
+
+ /*
+  * async queue for each priority case
+  */
+ struct cfq_queue *async_cfqq[2][IOPRIO_BE_NR];
+ struct cfq_queue *async_idle_cfqq;
+
+ sector_t last_position;
+ unsigned long last_end_request;
+
+ /*
+  * tunables, see top of file
+  */
+ unsigned int cfq_quantum;
+ unsigned int cfq_fifo_expire[2];
+ unsigned int cfq_back_penalty;
```

```
+ unsigned int cfq_back_max;
+ unsigned int cfq_slice[2];
+ unsigned int cfq_slice_async_rq;
+ unsigned int cfq_slice_idle;
+
+ struct list_head cic_list;
+};
+
+/*
+ * Per process-grouping structure
+ */
+struct cfq_queue {
+ /* reference count */
+ atomic_t ref;
+ /* parent cfq_data */
+ struct cfq_data *cfqd;
+ /* service_tree member */
+ struct rb_node rb_node;
+ /* service_tree key */
+ unsigned long rb_key;
+ /* sorted list of pending requests */
+ struct rb_root sort_list;
+ /* if fifo isn't expired, next request to serve */
+ struct request *next_rq;
+ /* requests queued in sort_list */
+ int queued[2];
+ /* currently allocated requests */
+ int allocated[2];
+ /* pending metadata requests */
+ int meta_pending;
+ /* fifo list of requests in sort_list */
+ struct list_head fifo;
+
+ unsigned long slice_end;
+ long slice_resid;
+
+ /* number of requests that are on the dispatch list or inside driver */
+ int dispatched;
+
+ /* io prio of this group */
+ unsigned short ioprio, org_ioprio;
+ unsigned short ioprio_class, org_ioprio_class;
+
+ /* various state flags, see below */
+ unsigned int flags;
+};
+
+#endif /* _LINUX_CFQ_IOSCHED_H */
```

```
--- linux-2.6.25-rc5-mm1/block/cfq-iosched.c.move 2008-02-15 00:49:11.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfq-iosched.c 2008-02-15 01:03:38.000000000 -0500
@@ -11,6 +11,7 @@
 #include <linux/elevator.h>
 #include <linux/rbtree.h>
 #include <linux/ioprio.h>
+#include <linux/cfq-iosched.h>

 /*
  * tunables
@@ -58,105 +59,6 @@ static struct completion *ioc_gone;

 #define sample_valid(samples) ((samples) > 80)

-/*
- * Most of our rbtree usage is for sorting with min extraction, so
- * if we cache the leftmost node we don't have to walk down the tree
- * to find it. Idea borrowed from Ingo Molnars CFS scheduler. We should
- * move this into the elevator for the rq sorting as well.
- */
-struct cfq_rb_root {
- struct rb_root rb;
- struct rb_node *left;
-};
-#define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }
-
-/*
- * Per block device queue structure
- */
-struct cfq_data {
- struct request_queue *queue;
-
- /*
-  * rr list of queues with requests and the count of them
-  */
- struct cfq_rb_root service_tree;
- unsigned int busy_queues;
-
- int rq_in_driver;
- int sync_flight;
- int hw_tag;
-
- /*
-  * idle window management
-  */
- struct timer_list idle_slice_timer;
- struct work_struct unplug_work;
-
```

```
- struct cfq_queue *active_queue;
- struct cfq_io_context *active_cic;
-
- /*
-  * async queue for each priority case
-  */
- struct cfq_queue *async_cfqq[2][IOPRIO_BE_NR];
- struct cfq_queue *async_idle_cfqq;
-
- sector_t last_position;
- unsigned long last_end_request;
-
- /*
-  * tunables, see top of file
-  */
- unsigned int cfq_quantum;
- unsigned int cfq_fifo_expire[2];
- unsigned int cfq_back_penalty;
- unsigned int cfq_back_max;
- unsigned int cfq_slice[2];
- unsigned int cfq_slice_async_rq;
- unsigned int cfq_slice_idle;
-
- struct list_head cic_list;
-};
-
-/*
- * Per process-grouping structure
- */
-struct cfq_queue {
- /* reference count */
- atomic_t ref;
- /* parent cfq_data */
- struct cfq_data *cfqd;
- /* service_tree member */
- struct rb_node rb_node;
- /* service_tree key */
- unsigned long rb_key;
- /* sorted list of pending requests */
- struct rb_root sort_list;
- /* if fifo isn't expired, next request to serve */
- struct request *next_rq;
- /* requests queued in sort_list */
- int queued[2];
- /* currently allocated requests */
- int allocated[2];
- /* pending metadata requests */
- int meta_pending;
```

```
- /* fifo list of requests in sort_list */
- struct list_head fifo;
-
- unsigned long slice_end;
- long slice_resid;
-
- /* number of requests that are on the dispatch list or inside driver */
- int dispatched;
-
- /* io prio of this group */
- unsigned short ioprio, org_ioprio;
- unsigned short ioprio_class, org_ioprio_class;
-
- /* various state flags, see below */
- unsigned int flags;
-};
-
 enum cfqq_state_flags {
  CFQ_CFQQ_FLAG_on_rr = 0, /* on round-robin busy list */
  CFQ_CFQQ_FLAG_wait_request, /* waiting for a request */
```

Subject: [RFC][PATCH 2/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:43 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Registers the cfqio_subsys subsystem in the CGroups framework.

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

```
--- /dev/null 2008-02-14 08:57:49.255923728 -0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h 2008-02-15 01:06:40.000000000 -0500
@@ -0,0 +1,26 @@
+/*
+ * include/linux/cfqio-cgroup.h
+ *
+ * cfqio_subsys: CGroup subsystem that allows CFQ recognize
+ * cgroups and perform scheduling according to this.
+ *
+ * Copyright (C) 2008 OpenVZ http://openvz.org
+ *
+ * Author: Vasily Tarasov <vtaras@openvz.org>
+ *
```

```
+ */
+
+#ifndef _LINUX_CFQIO_CGROUP_H
+#define _LINUX_CFQIO_CGROUP_H
+
+#define CFQIO_SS_IOPRIO_DEF 4
+#define CFQIO_SS_IOPRIO_MAX 7
+#define CFQIO_SS_IOPRIO_MIN 0
+
+/* cfqio_subsys's per cgroup state holder */
+struct cfqio_ss_css {
+ struct cgroup_subsys_state css;
+ unsigned int   ioprio;
+};
+
+#endif /* _LINUX_CFQIO_CGROUP_H */
--- linux-2.6.25-rc5-mm1/include/linux/cgroup_subsys.h.subsys 2008-02-15 00:49:56.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cgroup_subsys.h 2008-02-15 01:06:40.000000000 -0500
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_CFQIO
+SUBSYS(cfqio)
+#endif
+
+/* */
--- linux-2.6.25-rc5-mm1/block/Makefile.subsys 2008-02-15 00:49:11.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/Makefile 2008-02-15 01:06:40.000000000 -0500
@@ -11,6 +11,7 @@ obj-$(CONFIG_IOSCHED_NOOP) += noop-iosch
 obj-$(CONFIG_IOSCHED_AS) += as-iosched.o
 obj-$(CONFIG_IOSCHED_DEADLINE) += deadline-iosched.o
 obj-$(CONFIG_IOSCHED_CFQ) += cfq-iosched.o
+obj-$(CONFIG_CGROUP_CFQIO) += cfqio-cgroup.o

 obj-$(CONFIG_BLK_DEV_IO_TRACE) += blktrace.o
 obj-$(CONFIG_BLOCK_COMPAT) += compat_ioctl.o
--- /dev/null 2008-02-14 08:57:49.255923728 -0500
+++ linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c 2008-02-15 01:06:40.000000000 -0500
@@ -0,0 +1,97 @@
+/*
+ * block/cfqio-cgroup.c
+ *
+ * cfqio_cgroup: CGroup subsystem that allows CFQ recognize
+ * cgroups and perform scheduling according to this.
+ *
```

```
+ * Copyright (C) 2008 OpenVZ http://openvz.org
+ *
+ * Author: Vasily Tarasov <vtaras@openvz.org>
+ *
+ */
+
+#include <linux/cgroup.h>
+#include <linux/cfqio-cgroup.h>
+#include <linux/err.h>
+
+static void cfqio_ss_fini(struct cfqio_ss_css *cfqio_css)
+{
+}
+
+static void cfqio_ss_init(struct cfqio_ss_css *cfqio_css)
+{
+ cfqio_css->ioprio = CFQIO_SS_IOPRIO_DEF;
+}
+
+static struct cgroup_subsys_state *
+cfqio_ss_cgrp_create(struct cgroup_subsys *subsys, struct cgroup *cgrp)
+{
+ struct cfqio_ss_css *cfqio_css;
+
+ cfqio_css = kmalloc(sizeof(*cfqio_css), GFP_KERNEL);
+ if (!cfqio_css)
+  return ERR_PTR(-ENOMEM);
+
+ cfqio_ss_init(cfqio_css);
+
+ return &cfqio_css->css;
+}
+
+static inline struct cfqio_ss_css *
+cfqio_ss_cgrp_css(struct cgroup *cgrp)
+{
+ return container_of(cgrp->subsys[cfqio_subsys_id],
+     struct cfqio_ss_css, css);
+}
+
+static void
+cfqio_ss_cgrp_destroy(struct cgroup_subsys *subsys, struct cgroup *cgrp)
+{
+ struct cfqio_ss_css *cfqio_css;
+
+ cfqio_css = cfqio_ss_cgrp_css(cgrp);
+ cfqio_ss_fini(cfqio_css);
+ kfree(cfqio_css);
```

```
+ return;
+}
+
+static u64
+cfqio_ss_ioprio_read(struct cgroup *cgrp, struct cftype *cft)
+{
+ return (u64)cfqio_ss_cgrp_css(cgrp)->ioprio;
+}
+
+static int
+cfqio_ss_ioprio_write(struct cgroup *cgrp, struct cftype *cft, u64 val)
+{
+ if (val > CFQIO_SS_IOPRIO_MAX || val < CFQIO_SS_IOPRIO_MIN)
+  return -EINVAL;
+
+ cfqio_ss_cgrp_css(cgrp)->ioprio = val;
+
+ return 0;
+}
+
+/* array since more then one file are expected in the future */
+static struct cftype cfqio_ss_files[] =  {
+ {
+  .name =  "ioprio",
+  .read_uint = cfqio_ss_ioprio_read,
+  .write_uint = cfqio_ss_ioprio_write,
+ },
+};
+
+static int
+cfqio_ss_populate_dir(struct cgroup_subsys *ss, struct cgroup *cgrp)
+{
+ return cgroup_add_files(cgrp, ss, cfqio_ss_files,
+     ARRAY_SIZE(cfqio_ss_files));
+}
+
+struct cgroup_subsys cfqio_subsys = {
+ .name =  "cfqio_subsys",
+ .subsys_id = cfqio_subsys_id,
+ .create = cfqio_ss_cgrp_create,
+ .destroy = cfqio_ss_cgrp_destroy,
+ .populate = cfqio_ss_populate_dir,
+};
```

Subject: [RFC][PATCH 3/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ

Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:45 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Extends the original CFQ data sructures and adds the major cfqio_subsys
data structure: cfqio_cgroup_data. Adds several helper functions,
which will be called later from CFQ code to form proper data
structures interconnection.

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

```
--- linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h.mainstruct 2008-02-15 01:06:40.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h 2008-02-15 01:07:29.000000000 -0500
@@ -13,6 +13,10 @@
 #ifndef _LINUX_CFQIO_CGROUP_H
 #define _LINUX_CFQIO_CGROUP_H

+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/cfq-iosched.h>
+
 #define CFQIO_SS_IOPRIO_DEF 4
 #define CFQIO_SS_IOPRIO_MAX 7
 #define CFQIO_SS_IOPRIO_MIN 0
@@ -21,6 +25,31 @@
 struct cfqio_ss_css {
  struct cgroup_subsys_state css;
  unsigned int   ioprio;
+ struct list_head  cfqio_cgrp_head;
+ /* this lock protects the list above */
+ rwlock_t   cfqio_cgrp_lock;
+ /* list of all such objects, anchored at cfqio_ss_css_list */
+ struct list_head  list;
 };

+#ifdef CONFIG_CGROUP_CFQIO
+extern struct cfqio_cgroup_data *
+cfqio_cgrp_findcreate(struct cfqio_ss_css *, struct cfq_data *, gfp_t gfp_mask);
+extern void cfqio_ss_exit_queue(struct cfq_data *);
+#else
+static inline struct cfqio_cgroup_data *
+cfqio_cgrp_findcreate(struct cfqio_ss_css *cfqio_ss,
+   struct cfq_data *cfqd, gfp_t gfp_mask)
+{
+ return &cfqd->cfqio_cgroup;
```

```
+}
+
+extern void cfqio_ss_exit_queue(struct cfq_data *cfqd) { ; }
+#endif /* CONFIG_CGROUP_CFQIO */
+
+static inline void cfqio_init_cfqio_cgroup(struct cfqio_cgroup_data *cfqio_cgrp)
+{
+ cfqio_cgrp->service_tree = CFQ_RB_ROOT;
+}
+
 #endif /* _LINUX_CFQIO_CGROUP_H */
--- linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h.mainstruct 2008-02-15 01:03:38.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h 2008-02-15 01:07:29.000000000 -0500
@@ -14,11 +14,61 @@ struct cfq_rb_root {
 #define CFQ_RB_ROOT (struct cfq_rb_root) { RB_ROOT, NULL, }

 /*
+ * Each block device managed by CFQ I/O scheduler is represented
+ * by cfq_data structure. Certain members of this structure are
+ * moved to cfqio_cgroup_data on per-cgroup basis. Thus
+ * cfqio_cgroup_data structure is per (device, cgroup) pare.
+ *
+ * Cgroup holds a list head of all cfqio_croup_data, that belong to this
+ * cgroup, and cfq_data holds a list head of all active cfqio_cgroup_data
+ * for the device (active means that there are requests in-flight).
+ *
+ * Also cfqio_cgroup_data has a pointer to owning cgroup and cfq_data.
+ *
+ * For example, if there are two devices and three cgroups:
+ *
+ *   cfq_data 1  cfq_data 2
+ *       |       |
+ *       |       |
+ * cgroup 1 --- cfqio_cgroup_data ----- cfqio_cgroup_data
+ *       |       |
+ *       |       |
+ * cgroup 2 --- cfqio_cgroup_data ----- cfqio_cgroup_data
+ *       |       |
+ *       |       |
+ * cgroup 3 --- cfqio_cgroup_data ----- cfqio_cgroup_data
+ *
+ * One more basic CFQ scheduler data structure is cfq_queue,
+ * which is a queue of requests. For sync queues it's a per-process
+ * structure. While creating new cfq_queue we store cfqio_cgroup_data
+ * it belongs to, and later use this information in order to add
+ * the queue to proper lists.
+ *
```

```
+ * We can't place this structure to cfqio-cgroup.h because of include
+ * files circular dependency.
+ */
+struct cfqio_cgroup_data {
+ /* for cfqio_ss_css->cfqio_cgrp_head */
+ struct list_head cfqio_cgrp_list;
+ /* for cfqd->act_cfqio_cgrp_head */
+ struct list_head act_cfqio_cgrp_list;
+ struct cfq_data  *cfqd;
+ struct cfqio_ss_css *cfqio_css;
+ /* rr list of queues with requests */
+ struct cfq_rb_root service_tree;
+};
+
+/*
  * Per block device queue structure
  */
 struct cfq_data {
  struct request_queue *queue;

+#ifndef CONFIG_CGROUP_CFQIO
+ /* use this cgroup if CGROUP_CFQIO is off:
+    look at cfqio_cgrp_findcreate() */
+ struct cfqio_cgroup_data cfqio_cgroup;
+#endif
  /*
   * rr list of queues with requests and the count of them
   */
@@ -59,6 +109,11 @@ struct cfq_data {
  unsigned int cfq_slice_idle;

  struct list_head cic_list;
+
+ /* list of cgroups that have requests */
+ struct list_head act_cfqio_cgrp_head;
+ /* cgroup that owns a timeslice at the moment */
+ struct cfqio_cgroup_data *active_cfqio_cgroup;
 };

 /*
@@ -98,6 +153,9 @@ struct cfq_queue {

  /* various state flags, see below */
  unsigned int flags;
+
+ /* cgroup/device this queue belongs to */
+ struct cfqio_cgroup_data *cfqio_cgrp;
 };
```

```
 #endif /* _LINUX_CFQ_IOSCHED_H */
--- linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c.mainstruct 2008-02-15 01:06:40.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c 2008-02-15 01:07:29.000000000 -0500
@@ -10,17 +10,127 @@
  *
  */

+#include <linux/ioprio.h>
 #include <linux/cgroup.h>
 #include <linux/cfqio-cgroup.h>
 #include <linux/err.h>

+LIST_HEAD(cfqio_ss_css_head);
+/* This lock protects the list above.
+ * The global order of locking is the following:
+ * 1) queue_lock
+ * 2) cfqio_ss_css_locka
+ * 3) cfqio_ss_css->cfqio_cgrp_lock
+ */
+DEFINE_SPINLOCK(cfqio_ss_css_lock);
+
+static struct cfqio_cgroup_data *
+__find_cfqio_cgrp(struct cfqio_ss_css *cfqio_css, struct cfq_data *cfqd)
+{
+ struct cfqio_cgroup_data *cfqio_cgrp;
+
+ list_for_each_entry(cfqio_cgrp, &cfqio_css->cfqio_cgrp_head,
+       cfqio_cgrp_list)
+  if (cfqio_cgrp->cfqd == cfqd)
+   return cfqio_cgrp;
+
+ return NULL;
+}
+
+struct cfqio_cgroup_data *cfqio_cgrp_findcreate(struct cfqio_ss_css *cfqio_css,
+     struct cfq_data *cfqd, gfp_t gfp_mask)
+{
+ struct cfqio_cgroup_data *cfqio_cgrp_new;
+ struct cfqio_cgroup_data *cfqio_cgrp;
+
+ read_lock(&cfqio_css->cfqio_cgrp_lock);
+ cfqio_cgrp = __find_cfqio_cgrp(cfqio_css, cfqd);
+ read_unlock(&cfqio_css->cfqio_cgrp_lock);
+
+ if (cfqio_cgrp)
+  return cfqio_cgrp;
+
```

```
+ cfqio_cgrp_new = kzalloc(sizeof(*cfqio_cgrp_new), gfp_mask);
+ if (!cfqio_cgrp_new)
+  return NULL;
+
+ cfqio_init_cfqio_cgroup(cfqio_cgrp_new);
+ cfqio_cgrp_new->cfqd = cfqd;
+ cfqio_cgrp_new->cfqio_css = cfqio_css;
+
+ write_lock(&cfqio_css->cfqio_cgrp_lock);
+ cfqio_cgrp = __find_cfqio_cgrp(cfqio_css, cfqd);
+ if (cfqio_cgrp)
+  kfree(cfqio_cgrp_new);
+ else {
+  list_add_tail(&cfqio_cgrp_new->cfqio_cgrp_list,
+      &cfqio_css->cfqio_cgrp_head);
+  cfqio_cgrp = cfqio_cgrp_new;
+ }
+ write_unlock(&cfqio_css->cfqio_cgrp_lock);
+
+ return cfqio_cgrp;
+}
+
+static void release_cfqio_cgrp(struct cfqio_cgroup_data *cfqio_cgrp)
+{
+ list_del(&cfqio_cgrp->cfqio_cgrp_list);
+ kfree(cfqio_cgrp);
+}
+
+/* called on device queue exit */
+void cfqio_ss_exit_queue(struct cfq_data *cfqd)
+{
+ struct cfqio_ss_css *cfqio_css;
+ struct cfqio_cgroup_data *cfqio_cgrp;
+
+ spin_lock(&cfqio_ss_css_lock);
+ list_for_each_entry(cfqio_css, &cfqio_ss_css_head, list) {
+  write_lock(&cfqio_css->cfqio_cgrp_lock);
+  cfqio_cgrp = __find_cfqio_cgrp(cfqio_css, cfqd);
+  if (!cfqio_cgrp) {
+   write_unlock(&cfqio_css->cfqio_cgrp_lock);
+   continue;
+  }
+  release_cfqio_cgrp(cfqio_cgrp);
+  write_unlock(&cfqio_css->cfqio_cgrp_lock);
+ }
+ spin_unlock(&cfqio_ss_css_lock);
+}
+
```

```
+static void cfqio_ss_css_list_del(struct cfqio_ss_css *cfqio_css)
+{
+ spin_lock(&cfqio_ss_css_lock);
+ list_del(&cfqio_css->list);
+ spin_unlock(&cfqio_ss_css_lock);
+}
+
+static void cfqio_ss_css_list_add(struct cfqio_ss_css *cfqio_css)
+{
+ spin_lock(&cfqio_ss_css_lock);
+ list_add(&cfqio_css->list, &cfqio_ss_css_head);
+ spin_unlock(&cfqio_ss_css_lock);
+}
+
 static void cfqio_ss_fini(struct cfqio_ss_css *cfqio_css)
 {
+ struct cfqio_cgroup_data *cfqio_cgrp;
+ struct cfqio_cgroup_data *cfqio_cgrp_tmp;
+
+ cfqio_ss_css_list_del(cfqio_css);
+
+ /* no lock since cgroup is already dead */
+ list_for_each_entry_safe(cfqio_cgrp, cfqio_cgrp_tmp,
+   &cfqio_css->cfqio_cgrp_head, cfqio_cgrp_list)
+  release_cfqio_cgrp(cfqio_cgrp);
 }

 static void cfqio_ss_init(struct cfqio_ss_css *cfqio_css)
 {
  cfqio_css->ioprio = CFQIO_SS_IOPRIO_DEF;
+ INIT_LIST_HEAD(&cfqio_css->cfqio_cgrp_head);
+ rwlock_init(&cfqio_css->cfqio_cgrp_lock);
+ cfqio_ss_css_list_add(cfqio_css);
 }

 static struct cgroup_subsys_state *
```

Subject: [RFC][PATCH 4/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:47 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Adds hooks to CFQ code to form proper data structures interconnection.

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

```
---

--- linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h.creat-elim 2008-02-15 01:07:29.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h 2008-02-15 01:08:25.000000000 -0500
@@ -36,6 +36,12 @@ struct cfqio_ss_css {
 extern struct cfqio_cgroup_data *
 cfqio_cgrp_findcreate(struct cfqio_ss_css *, struct cfq_data *, gfp_t gfp_mask);
 extern void cfqio_ss_exit_queue(struct cfq_data *);
+
+static inline struct cfqio_ss_css *cfqio_css_by_tsk(struct task_struct *tsk)
+{
+ return container_of(tsk->cgroups->subsys[cfqio_subsys_id],
+      struct cfqio_ss_css, css);
+}
 #else
 static inline struct cfqio_cgroup_data *
 cfqio_cgrp_findcreate(struct cfqio_ss_css *cfqio_ss,
@@ -45,6 +51,11 @@ cfqio_cgrp_findcreate(struct cfqio_ss_cs
 }

 extern void cfqio_ss_exit_queue(struct cfq_data *cfqd) { ; }
+
+static inline struct cfqio_ss_css *cfqio_css_by_tsk(struct task_struct *tsk)
+{
+ return NULL;
+}
 #endif /* CONFIG_CGROUP_CFQIO */

 static inline void cfqio_init_cfqio_cgroup(struct cfqio_cgroup_data *cfqio_cgrp)
--- linux-2.6.25-rc5-mm1/block/cfq-iosched.c.creat-elim 2008-02-15 01:03:38.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfq-iosched.c 2008-02-15 01:08:25.000000000 -0500
@@ -12,6 +12,8 @@
 #include <linux/rbtree.h>
 #include <linux/ioprio.h>
 #include <linux/cfq-iosched.h>
+#include <linux/cgroup.h>
+#include <linux/cfqio-cgroup.h>

 /*
  * tunables
@@ -1256,11 +1258,14 @@ cfq_find_alloc_queue(struct cfq_data *cf
 {
  struct cfq_queue *cfqq, *new_cfqq = NULL;
  struct cfq_io_context *cic;
+ struct cfqio_cgroup_data *cfqio_cgrp = NULL;
+ struct cfqio_ss_css *cfqio_css;
```

```
 retry:
  cic = cfq_cic_lookup(cfqd, ioc);
  /* cic always exists here */
  cfqq = cic_to_cfqq(cic, is_sync);
+ cfqio_css = cfqio_css_by_tsk(current);

  if (!cfqq) {
   if (new_cfqq) {
@@ -1277,6 +1282,14 @@ retry:
    new_cfqq = kmem_cache_alloc_node(cfq_pool,
      gfp_mask | __GFP_NOFAIL | __GFP_ZERO,
      cfqd->queue->node);
+   if (new_cfqq) {
+    cfqio_cgrp = cfqio_cgrp_findcreate(cfqio_css,
+        cfqd, gfp_mask);
+    if (!cfqio_cgrp) {
+     kmem_cache_free(cfq_pool, new_cfqq);
+     new_cfqq = NULL;
+    }
+   }
    spin_lock_irq(cfqd->queue->queue_lock);
    goto retry;
   } else {
@@ -1285,6 +1298,13 @@ retry:
      cfqd->queue->node);
    if (!cfqq)
     goto out;
+
+   cfqio_cgrp = cfqio_cgrp_findcreate(cfqio_css,
+       cfqd, gfp_mask);
+   if (!cfqio_cgrp) {
+    kmem_cache_free(cfq_pool, cfqq);
+    cfqq = NULL;
+   }
   }

   RB_CLEAR_NODE(&cfqq->rb_node);
@@ -1298,6 +1318,8 @@ retry:

   cfq_init_prio_data(cfqq, ioc);

+  cfqq->cfqio_cgrp = cfqio_cgrp;
+
   if (is_sync) {
    if (!cfq_class_idle(cfqq))
     cfq_mark_cfqq_idle_window(cfqq);
@@ -1990,6 +2012,8 @@ static void cfq_exit_queue(elevator_t *e
```

```
  cfq_shutdown_timer_wq(cfqd);

+ cfqio_ss_exit_queue(cfqd);
+
  kfree(cfqd);
 }

@@ -2002,6 +2026,9 @@ static void *cfq_init_queue(struct reque
   return NULL;

  cfqd->service_tree = CFQ_RB_ROOT;
+#ifndef CONFIG_CGROUP_CFQIO
+ cfqio_init_cfqio_cgroup(&cfqd->cfqio_cgroup);
+#endif
  INIT_LIST_HEAD(&cfqd->cic_list);

  cfqd->queue = q;
@@ -2022,6 +2049,7 @@ static void *cfq_init_queue(struct reque
  cfqd->cfq_slice[1] = cfq_slice_sync;
  cfqd->cfq_slice_async_rq = cfq_slice_async_rq;
  cfqd->cfq_slice_idle = cfq_slice_idle;
+ INIT_LIST_HEAD(&cfqd->act_cfqio_cgrp_head);

  return cfqd;
 }
```

Subject: [RFC][PATCH 5/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:49 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Switches the CFQ scheduler to use per-cgroup queue tree,
instead one queue tree per device.

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

```
--- linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h.switch 2008-02-15 01:07:29.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h 2008-02-15 01:09:09.000000000 -0500
@@ -70,9 +70,8 @@ struct cfq_data {
  struct cfqio_cgroup_data cfqio_cgroup;
 #endif
```

```
  /*
-  * rr list of queues with requests and the count of them
+  * count of queues
   */
- struct cfq_rb_root service_tree;
  unsigned int busy_queues;

  int rq_in_driver;
--- linux-2.6.25-rc5-mm1/block/cfq-iosched.c.switch 2008-02-15 01:08:25.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfq-iosched.c 2008-02-15 01:09:09.000000000 -0500
@@ -354,6 +354,7 @@ static unsigned long cfq_slice_offset(st
 static void cfq_service_tree_add(struct cfq_data *cfqd,
        struct cfq_queue *cfqq, int add_front)
 {
+ struct cfqio_cgroup_data *cfqio_cgrp = cfqq->cfqio_cgrp;
  struct rb_node **p, *parent;
  struct cfq_queue *__cfqq;
  unsigned long rb_key;
@@ -361,7 +362,7 @@ static void cfq_service_tree_add(struct

  if (cfq_class_idle(cfqq)) {
   rb_key = CFQ_IDLE_DELAY;
-  parent = rb_last(&cfqd->service_tree.rb);
+  parent = rb_last(&cfqio_cgrp->service_tree.rb);
   if (parent && parent != &cfqq->rb_node) {
    __cfqq = rb_entry(parent, struct cfq_queue, rb_node);
    rb_key += __cfqq->rb_key;
@@ -381,12 +382,12 @@ static void cfq_service_tree_add(struct
   if (rb_key == cfqq->rb_key)
    return;

-  cfq_rb_erase(&cfqq->rb_node, &cfqd->service_tree);
+  cfq_rb_erase(&cfqq->rb_node, &cfqio_cgrp->service_tree);
  }

  left = 1;
  parent = NULL;
- p = &cfqd->service_tree.rb.rb_node;
+ p = &cfqio_cgrp->service_tree.rb.rb_node;
  while (*p) {
   struct rb_node **n;

@@ -418,11 +419,11 @@ static void cfq_service_tree_add(struct
  }

  if (left)
-  cfqd->service_tree.left = &cfqq->rb_node;
+  cfqio_cgrp->service_tree.left = &cfqq->rb_node;
```

```
  cfqq->rb_key = rb_key;
  rb_link_node(&cfqq->rb_node, parent, p);
- rb_insert_color(&cfqq->rb_node, &cfqd->service_tree.rb);
+ rb_insert_color(&cfqq->rb_node, &cfqio_cgrp->service_tree.rb);
 }

 /*
@@ -456,11 +457,12 @@ static void cfq_add_cfqq_rr(struct cfq_d
  */
 static void cfq_del_cfqq_rr(struct cfq_data *cfqd, struct cfq_queue *cfqq)
 {
+ struct cfqio_cgroup_data *cfqio_cgrp = cfqq->cfqio_cgrp;
  BUG_ON(!cfq_cfqq_on_rr(cfqq));
  cfq_clear_cfqq_on_rr(cfqq);

  if (!RB_EMPTY_NODE(&cfqq->rb_node))
-  cfq_rb_erase(&cfqq->rb_node, &cfqd->service_tree);
+  cfq_rb_erase(&cfqq->rb_node, &cfqio_cgrp->service_tree);

  BUG_ON(!cfqd->busy_queues);
  cfqd->busy_queues--;
@@ -704,10 +706,16 @@ static inline void cfq_slice_expired(str
  */
 static struct cfq_queue *cfq_get_next_queue(struct cfq_data *cfqd)
 {
- if (RB_EMPTY_ROOT(&cfqd->service_tree.rb))
+ struct cfqio_cgroup_data *cfqio_cgrp;
+
+ cfqio_cgrp = cfqd->active_cfqio_cgroup;
+ if (!cfqio_cgrp)
+  return NULL;
+
+ if (RB_EMPTY_ROOT(&cfqio_cgrp->service_tree.rb))
   return NULL;

- return cfq_rb_first(&cfqd->service_tree);
+ return cfq_rb_first(&cfqio_cgrp->service_tree);
 }

 /*
@@ -964,7 +972,7 @@ static int __cfq_forced_dispatch_cfqq(st
  * Drain our current requests. Used for barriers and when switching
  * io schedulers on-the-fly.
  */
-static int cfq_forced_dispatch(struct cfq_data *cfqd)
+static int __cfq_forced_dispatch(struct cfqio_cgroup_data *cfqd)
 {
```

```
  struct cfq_queue *cfqq;
  int dispatched = 0;
@@ -972,6 +980,25 @@ static int cfq_forced_dispatch(struct cf
  while ((cfqq = cfq_rb_first(&cfqd->service_tree)) != NULL)
   dispatched += __cfq_forced_dispatch_cfqq(cfqq);

+ return dispatched;
+}
+
+static int cfq_forced_dispatch(struct cfq_data *cfqd)
+{
+ struct cfqio_cgroup_data *cfqio_cgrp;
+ struct cfqio_cgroup_data *cfqio_cgrp_tmp;
+ int dispatched;
+
+ dispatched = 0;
+
+ /*
+  * We use _safe iterating here because  __cfq_forced_dispatch()
+  * produces list_del() implicitly
+  */
+ list_for_each_entry_safe(cfqio_cgrp, cfqio_cgrp_tmp,
+    &cfqd->act_cfqio_cgrp_head, act_cfqio_cgrp_list)
+  dispatched += __cfq_forced_dispatch(cfqio_cgrp);
+
  cfq_slice_expired(cfqd, 0);

  BUG_ON(cfqd->busy_queues);
@@ -2025,7 +2052,6 @@ static void *cfq_init_queue(struct reque
  if (!cfqd)
   return NULL;

- cfqd->service_tree = CFQ_RB_ROOT;
 #ifndef CONFIG_CGROUP_CFQIO
  cfqio_init_cfqio_cgroup(&cfqd->cfqio_cgroup);
 #endif
```

Subject: [RFC][PATCH 6/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:51 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Introduces cgroups scheduling. Each time when I/O request is placed on per-process request queue and there were no other requests from this cgroup, this cgroup is added to the end of active cgroups list.

This list is service in round-robin fashion. Switching between
cgroups happens eiher when cgroup expires its time slice, either if there
are no more requests from it. Each time I/O request is completed,
we check if it was the last request from cgroup, and in this case remove
cgroup from the active list.

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

```
--- linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h.cgrpsched 2008-02-15 01:08:25.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfqio-cgroup.h 2008-02-15 01:10:42.000000000 -0500
@@ -42,6 +42,10 @@ static inline struct cfqio_ss_css *cfqio
  return container_of(tsk->cgroups->subsys[cfqio_subsys_id],
       struct cfqio_ss_css, css);
 }
+extern int cfqio_cgrp_expired(struct cfq_data *);
+extern void cfqio_cgrp_schedule_active(struct cfq_data *);
+extern void  cfqio_cgrp_inc_rqnum(struct cfq_queue *);
+extern void cfqio_cgrp_dec_rqnum(struct cfq_queue *);
 #else
 static inline struct cfqio_cgroup_data *
 cfqio_cgrp_findcreate(struct cfqio_ss_css *cfqio_ss,
@@ -56,6 +60,16 @@ static inline struct cfqio_ss_css *cfqio
 {
  return NULL;
 }
+
+static inline int cfqio_cgrp_expired(struct cfq_data *cfqd) { return 0; }
+
+static inline void cfqio_cgrp_schedule_active(struct cfq_data *cfqd)
+{
+ cfqd->active_cfqio_cgroup = &cfqd->cfqio_cgroup;
+}
+
+static inline void cfqio_cgrp_inc_rqnum(struct cfq_queue *cfqq) { ; }
+static inline void cfqio_cgrp_dec_rqnum(struct cfq_queue *cfqq) { ; }
 #endif /* CONFIG_CGROUP_CFQIO */

 static inline void cfqio_init_cfqio_cgroup(struct cfqio_cgroup_data *cfqio_cgrp)
--- linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h.cgrpsched 2008-02-15 01:09:09.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h 2008-02-15 01:10:42.000000000 -0500
@@ -56,6 +56,7 @@ struct cfqio_cgroup_data {
  struct cfqio_ss_css *cfqio_css;
  /* rr list of queues with requests */
  struct cfq_rb_root service_tree;
```

```
+ unsigned long  rqnum;
 };

 /*
@@ -113,6 +114,8 @@ struct cfq_data {
  struct list_head act_cfqio_cgrp_head;
  /* cgroup that owns a timeslice at the moment */
  struct cfqio_cgroup_data *active_cfqio_cgroup;
+ unsigned int cfqio_cgrp_slice;
+ unsigned long cfqio_slice_end;
 };

 /*
--- linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c.cgrpsched 2008-02-15 01:07:29.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c 2008-02-15 01:10:42.000000000 -0500
@@ -24,6 +24,65 @@ LIST_HEAD(cfqio_ss_css_head);
  */
 DEFINE_SPINLOCK(cfqio_ss_css_lock);

+int cfqio_cgrp_expired(struct cfq_data *cfqd)
+{
+ return time_after(jiffies, cfqd->cfqio_slice_end) ?  1 : 0;
+}
+
+static inline unsigned long time_slice_by_ioprio(unsigned int ioprio,
+       unsigned int base_slice)
+{
+ return  base_slice +
+  (base_slice * (ioprio - CFQIO_SS_IOPRIO_MIN))
+   / (CFQIO_SS_IOPRIO_MAX - CFQIO_SS_IOPRIO_MIN);
+}
+
+static inline void set_active_cgrp(struct cfq_data *cfqd)
+{
+ if (list_empty(&cfqd->act_cfqio_cgrp_head))
+  return;
+
+ cfqd->active_cfqio_cgroup = list_first_entry(&cfqd->act_cfqio_cgrp_head,
+    struct cfqio_cgroup_data, act_cfqio_cgrp_list);
+ list_move_tail(&cfqd->active_cfqio_cgroup->act_cfqio_cgrp_list,
+     &cfqd->act_cfqio_cgrp_head);
+ cfqd->cfqio_slice_end = jiffies +
+ time_slice_by_ioprio(cfqd->active_cfqio_cgroup->cfqio_css->ioprio,
+       cfqd->cfqio_cgrp_slice);
+}
+
+void cfqio_cgrp_schedule_active(struct cfq_data *cfqd)
+{
```

```
+ if (cfqio_cgrp_expired(cfqd) || !cfqd->active_cfqio_cgroup ||
+    !cfqd->active_cfqio_cgroup->rqnum)
+  set_active_cgrp(cfqd);
+}
+
+void cfqio_cgrp_inc_rqnum(struct cfq_queue *cfqq)
+{
+ struct cfqio_cgroup_data *cfqio_cgrp;
+
+ cfqio_cgrp = cfqq->cfqio_cgrp;
+
+ if (!cfqio_cgrp->rqnum)
+  list_add_tail(&cfqio_cgrp->act_cfqio_cgrp_list,
+    &cfqq->cfqd->act_cfqio_cgrp_head);
+
+ cfqio_cgrp->rqnum++;
+}
+
+void cfqio_cgrp_dec_rqnum(struct cfq_queue *cfqq)
+{
+ struct cfqio_cgroup_data *cfqio_cgrp;
+
+ cfqio_cgrp = cfqq->cfqio_cgrp;
+
+ cfqio_cgrp->rqnum--;
+
+ if (!cfqio_cgrp->rqnum)
+  list_del(&cfqio_cgrp->act_cfqio_cgrp_list);
+}
+
 static struct cfqio_cgroup_data *
 __find_cfqio_cgrp(struct cfqio_ss_css *cfqio_css, struct cfq_data *cfqd)
 {
--- linux-2.6.25-rc5-mm1/block/cfq-iosched.c.cgrpsched 2008-02-15 01:09:09.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfq-iosched.c 2008-02-15 01:10:42.000000000 -0500
@@ -29,6 +29,7 @@ static const int cfq_slice_sync = HZ / 1
 static int cfq_slice_async = HZ / 25;
 static const int cfq_slice_async_rq = 2;
 static int cfq_slice_idle = HZ / 125;
+static int cfqio_cgrp_slice = HZ / 2;

 /*
  * offset from end of service tree
@@ -185,6 +186,8 @@ static inline int cfq_slice_used(struct
 {
  if (cfq_cfqq_slice_new(cfqq))
   return 0;
+ if (cfqio_cgrp_expired(cfqq->cfqd))
```

```
+  return 1;
  if (time_before(jiffies, cfqq->slice_end))
   return 0;

@@ -447,6 +450,7 @@ static void cfq_add_cfqq_rr(struct cfq_d
  BUG_ON(cfq_cfqq_on_rr(cfqq));
  cfq_mark_cfqq_on_rr(cfqq);
  cfqd->busy_queues++;
+ cfqio_cgrp_inc_rqnum(cfqq);

  cfq_resort_rr_list(cfqd, cfqq);
 }
@@ -466,6 +470,7 @@ static void cfq_del_cfqq_rr(struct cfq_d

  BUG_ON(!cfqd->busy_queues);
  cfqd->busy_queues--;
+ cfqio_cgrp_dec_rqnum(cfqq);
 }

 /*
@@ -708,6 +713,8 @@ static struct cfq_queue *cfq_get_next_qu
 {
  struct cfqio_cgroup_data *cfqio_cgrp;

+ cfqio_cgrp_schedule_active(cfqd);
+
  cfqio_cgrp = cfqd->active_cfqio_cgroup;
  if (!cfqio_cgrp)
   return NULL;
@@ -2076,6 +2083,7 @@ static void *cfq_init_queue(struct reque
  cfqd->cfq_slice_async_rq = cfq_slice_async_rq;
  cfqd->cfq_slice_idle = cfq_slice_idle;
  INIT_LIST_HEAD(&cfqd->act_cfqio_cgrp_head);
+ cfqd->cfqio_cgrp_slice = cfqio_cgrp_slice;

  return cfqd;
 }
```

Subject: [RFC][PATCH 7/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:53 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Takes into account requests that are in driver now. Before that we switched
cgroup if there were no requests to service in this cgroup. Now we also

check that there are no requests from this cgroups currently processed by
driver and only in this case allow switching.

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

```
--- linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h.rqindrv 2008-02-15 01:10:42.000000000
-0500
+++ linux-2.6.25-rc5-mm1/include/linux/cfq-iosched.h 2008-02-15 01:11:45.000000000 -0500
@@ -57,6 +57,7 @@ struct cfqio_cgroup_data {
  /* rr list of queues with requests */
  struct cfq_rb_root service_tree;
  unsigned long  rqnum;
+ unsigned long  on_dispatch;
 };

 /*
--- linux-2.6.25-rc5-mm1/block/cfq-iosched.c.rqindrv 2008-02-15 01:10:42.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfq-iosched.c 2008-02-15 01:11:45.000000000 -0500
@@ -823,6 +823,7 @@ static void cfq_dispatch_insert(struct r

  cfq_remove_request(rq);
  cfqq->dispatched++;
+ cfqq->cfqio_cgrp->on_dispatch++;
  elv_dispatch_sort(q, rq);

  if (cfq_cfqq_sync(cfqq))
@@ -1775,6 +1776,7 @@ static void cfq_completed_request(struct
  WARN_ON(!cfqq->dispatched);
  cfqd->rq_in_driver--;
  cfqq->dispatched--;
+ cfqq->cfqio_cgrp->on_dispatch--;

  if (cfq_cfqq_sync(cfqq))
   cfqd->sync_flight--;
--- linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c.rqindrv 2008-02-15 01:10:42.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfqio-cgroup.c 2008-02-15 01:11:45.000000000 -0500
@@ -54,7 +54,8 @@ static inline void set_active_cgrp(struc
 void cfqio_cgrp_schedule_active(struct cfq_data *cfqd)
 {
  if (cfqio_cgrp_expired(cfqd) || !cfqd->active_cfqio_cgroup ||
-    !cfqd->active_cfqio_cgroup->rqnum)
+  (!cfqd->active_cfqio_cgroup->rqnum &&
+   !cfqd->active_cfqio_cgroup->on_dispatch))
   set_active_cgrp(cfqd);
 }
```

Subject: [RFC][PATCH 8/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:55 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Configurable cgroup time slice support.

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

```
--- linux-2.6.25-rc5-mm1/block/cfq-iosched.c.slice 2008-02-15 01:11:45.000000000 -0500
+++ linux-2.6.25-rc5-mm1/block/cfq-iosched.c 2008-02-15 01:12:22.000000000 -0500
@@ -2150,6 +2150,7 @@ SHOW_FUNCTION(cfq_slice_idle_show, cfqd-
 SHOW_FUNCTION(cfq_slice_sync_show, cfqd->cfq_slice[1], 1);
 SHOW_FUNCTION(cfq_slice_async_show, cfqd->cfq_slice[0], 1);
 SHOW_FUNCTION(cfq_slice_async_rq_show, cfqd->cfq_slice_async_rq, 0);
+SHOW_FUNCTION(cfq_cgrp_slice_show, cfqd->cfqio_cgrp_slice, 1);
 #undef SHOW_FUNCTION

 #define STORE_FUNCTION(__FUNC, __PTR, MIN, MAX, __CONV)   \
@@ -2181,6 +2182,7 @@ STORE_FUNCTION(cfq_slice_sync_store, &cf
 STORE_FUNCTION(cfq_slice_async_store, &cfqd->cfq_slice[0], 1, UINT_MAX, 1);
 STORE_FUNCTION(cfq_slice_async_rq_store, &cfqd->cfq_slice_async_rq, 1,
   UINT_MAX, 0);
+STORE_FUNCTION(cfq_cgrp_slice_store, &cfqd->cfqio_cgrp_slice, 1, UINT_MAX, 1);
 #undef STORE_FUNCTION

 #define CFQ_ATTR(name) \
@@ -2196,6 +2198,7 @@ static struct elv_fs_entry cfq_attrs[] =
  CFQ_ATTR(slice_async),
  CFQ_ATTR(slice_async_rq),
  CFQ_ATTR(slice_idle),
+ CFQ_ATTR(cgrp_slice),
  __ATTR_NULL
 };
```

Subject: [RFC][PATCH 9/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on CFQ
Posted by Vasily Tarasov on Fri, 15 Feb 2008 06:59:58 GMT
View Forum Message <> Reply to Message

From: Vasily Tarasov <vtaras@openvz.org>

Adds the description to the Kconfig file.

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

```
--- linux-2.6.25-rc5-mm1/init/Kconfig.kconfig 2008-02-15 01:01:40.000000000 -0500
+++ linux-2.6.25-rc5-mm1/init/Kconfig 2008-02-15 01:13:12.000000000 -0500
@@ -316,6 +316,13 @@ config GROUP_SCHED
   This feature lets CPU scheduler recognize task groups and control CPU
   bandwidth allocation to such task groups.

+config CGROUP_CFQIO
+ bool "CFQIO cgroup subsystem"
+ depends on CGROUPS && IOSCHED_CFQ=y
+ help
+   This feature allows CFQ I/O scheduler recognize cgroups and
+   control I/O bandwidth allocation to cgroups.
+
 config FAIR_GROUP_SCHED
  bool "Group scheduling for SCHED_OTHER"
  depends on GROUP_SCHED
```

Subject: Re: [RFC][PATCH 0/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based on C
Posted by Paul Jackson on Wed, 02 Apr 2008 05:31:27 GMT
View Forum Message <> Reply to Message

A couple of details:
1) Your clock is way off. These patches have a date stamp of Feb 15, but were actually sent out March 21 (and I'm behind in my reading of lkml - sorry ;).
2) Please put a different Subject on each path ... not just the N/M patch number, but use different words to describe each patch.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214