Subject: [PATCH 0/2] dm-ioband: I/O bandwidth controller v1.0.0: Introduction
Posted by Ryo Tsuruta on Mon, 19 May 2008 08:22:38 GMT
View Forum Message <> Reply to Message

Hi everyone,

This is dm-ioband version 1.0.0 release.

Dm-ioband is an I/O bandwidth controller implemented as a device-mapper
driver, which gives specified bandwidth to each job running on the same
physical device.

- Can be applied to the kernel 2.6.26-rc2-mm1.
- Changes from 0.0.4 (24th April):
  - Performance tuning
    - A new bandwidth control policy is added. This policy controls
      bandwidth based on the number of I/O sectors instead of the
 number of I/O requests.
 I think it would be useful for some applications such as video
 streaming or when you want to use high-end storages.
      - Control read and write requests separately. So even if a
        write I/O queue in an elevator is full, read I/O requests
  won't be blocked.
    - More accurate bandwidth control.
  - Code cleanups. Remove the "LINUX_VERSION_CODE" and "inline" stuff.
  - Fix panic on "dmsetup suspend."

For more details, please refer to:
http://people.valinux.co.jp/~ryov/dm-ioband/

Thanks,
Ryo Tsuruta

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

Subject: [PATCH 1/2] dm-ioband: I/O bandwidth controller v1.0.0: Source code and patch
Posted by Ryo Tsuruta on Mon, 19 May 2008 08:23:34 GMT
View Forum Message <> Reply to Message

Here is the patch of dm-ioband.

Based on 2.6.26-rc2-mm1
Signed-off-by: Ryo Tsuruta <ryov@valinux.co.jp>
Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```
diff -uprN linux-2.6.26-rc2-mm1.orig/drivers/md/Kconfig linux-2.6.26-rc2-mm1/drivers/md/Kconfig
--- linux-2.6.26-rc2-mm1.orig/drivers/md/Kconfig 2008-05-16 16:20:23.000000000 +0900
+++ linux-2.6.26-rc2-mm1/drivers/md/Kconfig 2008-05-19 14:22:37.000000000 +0900
@@ -271,4 +271,17 @@ config DM_UEVENT
  ---help---
  Generate udev events for DM events.

+config DM_IOBAND
+ tristate "I/O bandwidth control (EXPERIMENTAL)"
+ depends on BLK_DEV_DM && EXPERIMENTAL
+ ---help---
+ This device-mapper target allows to define how the
+ available bandwith of a storage device should be
+ shared between processes, cgroups, the partitions or the LUNs.
+
+ Information on how to use dm-ioband is available in:
+    <file:Documentation/device-mapper/ioband.txt>.
+
+ If unsure, say N.
+
 endif # MD
diff -uprN linux-2.6.26-rc2-mm1.orig/drivers/md/Makefile linux-2.6.26-rc2-mm1/drivers/md/Makefile
--- linux-2.6.26-rc2-mm1.orig/drivers/md/Makefile 2008-05-16 16:20:23.000000000 +0900
+++ linux-2.6.26-rc2-mm1/drivers/md/Makefile 2008-05-19 14:22:37.000000000 +0900
@@ -7,6 +7,7 @@ dm-mod-objs := dm.o dm-table.o dm-target
 dm-multipath-objs := dm-path-selector.o dm-mpath.o
 dm-snapshot-objs := dm-snap.o dm-exception-store.o
 dm-mirror-objs := dm-raid1.o
+dm-ioband-objs := dm-ioband-ctl.o dm-ioband-policy.o dm-ioband-type.o
 md-mod-objs     := md.o bitmap.o
 raid456-objs := raid5.o raid6algos.o raid6recov.o raid6tables.o \
      raid6int1.o raid6int2.o raid6int4.o \
@@ -36,6 +37,7 @@ obj-$(CONFIG_DM_MULTIPATH) += dm-multipa
 obj-$(CONFIG_DM_SNAPSHOT) += dm-snapshot.o
 obj-$(CONFIG_DM_MIRROR)  += dm-mirror.o dm-log.o
 obj-$(CONFIG_DM_ZERO)  += dm-zero.o
+obj-$(CONFIG_DM_IOBAND)  += dm-ioband.o

 quiet_cmd_unroll = UNROLL  $@
       cmd_unroll = $(PERL) $(srctree)/$(src)/unroll.pl $(UNROLL) \
diff -uprN linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband-ctl.c
linux-2.6.26-rc2-mm1/drivers/md/dm-ioband-ctl.c
--- linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband-ctl.c 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.26-rc2-mm1/drivers/md/dm-ioband-ctl.c 2008-05-19 14:22:37.000000000 +0900
@@ -0,0 +1,1108 @@
+/*
+ * Copyright (C) 2008 VA Linux Systems Japan K.K.
```

```
+ * Authors: Hirokazu Takahashi <taka@valinux.co.jp>
+ *          Ryo Tsuruta <ryov@valinux.co.jp>
+ *
+ *  I/O bandwidth control
+ *
+ * This file is released under the GPL.
+ */
+#include <linux/module.h>
+#include <linux/init.h>
+#include <linux/bio.h>
+#include <linux/slab.h>
+#include <linux/workqueue.h>
+#include <linux/raid/md.h>
+#include "dm.h"
+#include "dm-bio-list.h"
+#include "dm-ioband.h"
+
+#define DM_MSG_PREFIX "ioband"
+#define POLICY_PARAM_START 6
+#define POLICY_PARAM_DELIM "=:,"
+
+static LIST_HEAD(ioband_device_list);
+/* to protect ioband_device_list */
+static DEFINE_SPINLOCK(ioband_devicelist_lock);
+
+static void ioband_conduct(struct work_struct *);
+static void ioband_hold_bio(struct ioband_group *, struct bio *);
+static struct bio *ioband_pop_bio(struct ioband_group *);
+static int ioband_set_param(struct ioband_group *, char *, char *);
+static int ioband_group_attach(struct ioband_group *, int, char *);
+static int ioband_group_type_select(struct ioband_group *, char *);
+
+long ioband_debug; /* just for debugging */
+
+static int policy_init(struct ioband_device *dp, char *name,
+       int argc, char **argv)
+{
+ struct policy_type *p;
+ int r;
+
+ for (p = dm_ioband_policy_type; (p->p_name); p++) {
+  if (!strcmp(name, p->p_name))
+   break;
+ }
+
+ dp->g_policy = p;
+ r = p->p_policy_init(dp, argc, argv);
+ if (!dp->g_hold_bio)
```

```
+  dp->g_hold_bio = ioband_hold_bio;
+ if (!dp->g_pop_bio)
+  dp->g_pop_bio = ioband_pop_bio;
+ return r;
+}
+
+static struct ioband_device *alloc_ioband_device(char *name, char *policy,
+   int io_throttle, int io_limit, int argc, char **argv)
+
+{
+ struct ioband_device *dp = NULL;
+ struct ioband_device *p;
+ struct ioband_device *new;
+ unsigned long flags;
+
+ new = kzalloc(sizeof(struct ioband_device), GFP_KERNEL);
+ if (!new)
+  goto try_to_find;
+
+ /*
+  * Prepare its own workqueue as generic_make_request() may potentially
+  * block the workqueue when submitting BIOs.
+  */
+ new->g_ioband_wq = create_workqueue("kioband");
+ if (!new->g_ioband_wq) {
+  kfree(new);
+  new = NULL;
+  goto try_to_find;
+ }
+
+ INIT_WORK(&new->g_conductor, ioband_conduct);
+ INIT_LIST_HEAD(&new->g_groups);
+ INIT_LIST_HEAD(&new->g_list);
+ spin_lock_init(&new->g_lock);
+ new->g_io_throttle = io_throttle;
+ new->g_io_limit[0] = io_limit;
+ new->g_io_limit[1] = io_limit;
+ new->g_issued[0] = 0;
+ new->g_issued[1] = 0;
+ new->g_blocked = 0;
+ new->g_ref = 0;
+ new->g_flags = 0;
+ strlcpy(new->g_name, name, sizeof(new->g_name));
+ new->g_policy = NULL;
+ new->g_hold_bio = NULL;
+ new->g_pop_bio = NULL;
+ init_waitqueue_head(&new->g_waitq);
+
```

```
+try_to_find:
+ spin_lock_irqsave(&ioband_devicelist_lock, flags);
+ list_for_each_entry(p, &ioband_device_list, g_list) {
+  if (!strcmp(p->g_name, name)) {
+   dp = p;
+   break;
+  }
+ }
+ if (!dp && (new)) {
+  if (policy_init(new, policy, argc, argv) == 0) {
+   dp = new;
+   new = NULL;
+   list_add_tail(&dp->g_list, &ioband_device_list);
+  }
+ }
+ spin_unlock_irqrestore(&ioband_devicelist_lock, flags);
+
+ if (new) {
+  destroy_workqueue(new->g_ioband_wq);
+  kfree(new);
+ }
+
+ return dp;
+}
+
+static void release_ioband_device(struct ioband_device *dp)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&ioband_devicelist_lock, flags);
+ if (!list_empty(&dp->g_groups)) {
+  spin_unlock_irqrestore(&ioband_devicelist_lock, flags);
+  return;
+ }
+ list_del(&dp->g_list);
+ spin_unlock_irqrestore(&ioband_devicelist_lock, flags);
+ destroy_workqueue(dp->g_ioband_wq);
+ kfree(dp);
+}
+
+static struct ioband_group *ioband_group_find(struct ioband_group *head,
+         int id)
+{
+ struct ioband_group *p;
+ struct ioband_group *gp = NULL;
+
+ list_for_each_entry(p, &head->c_group_list, c_group_list) {
+  if (p->c_id == id || id == IOBAND_ID_ANY)
```

```
+   gp = p;
+ }
+ return gp;
+}
+
+static int ioband_group_init(struct ioband_group *gp,
+     struct ioband_group *head, struct ioband_device *dp, int id, char *param)
+{
+ unsigned long flags;
+ int r;
+
+ INIT_LIST_HEAD(&gp->c_list);
+ bio_list_init(&gp->c_blocked_bios);
+ bio_list_init(&gp->c_prio_bios);
+ gp->c_id = id; /* should be verified */
+ gp->c_blocked = 0;
+ gp->c_prio_blocked = 0;
+ memset(gp->c_stat, 0, sizeof(gp->c_stat));
+ init_waitqueue_head(&gp->c_waitq);
+ gp->c_flags = 0;
+
+ INIT_LIST_HEAD(&gp->c_group_list);
+
+ gp->c_banddev = dp;
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+ if (head && ioband_group_find(head, id)) {
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  DMWARN("ioband_group: id=%d already exists.", id);
+  return -EEXIST;
+ }
+
+ dp->g_ref++;
+ list_add_tail(&gp->c_list, &dp->g_groups);
+
+ r = dp->g_group_ctr(gp, param);
+ if (r) {
+  list_del(&gp->c_list);
+  dp->g_ref--;
+  return r;
+ }
+
+ if (head) {
+  list_add_tail(&gp->c_group_list, &head->c_group_list);
+  gp->c_dev = head->c_dev;
+  gp->c_target = head->c_target;
+ }
+
```

```
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+
+ return 0;
+}
+
+static void ioband_group_release(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ list_del(&gp->c_list);
+ list_del(&gp->c_group_list);
+ dp->g_ref--;
+ dp->g_group_dtr(gp);
+ kfree(gp);
+}
+
+static void ioband_group_destroy_all(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct ioband_group *group;
+ unsigned long flags;
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+ while ((group = ioband_group_find(gp, IOBAND_ID_ANY)))
+  ioband_group_release(group);
+ ioband_group_release(gp);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+}
+
+static void ioband_group_stop(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ unsigned long flags;
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+ set_group_down(gp);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+ queue_work(dp->g_ioband_wq, &dp->g_conductor);
+ flush_workqueue(dp->g_ioband_wq);
+}
+
+static void ioband_group_stop_all(struct ioband_group *head, int suspend)
+{
+ struct ioband_device *dp = head->c_banddev;
+ struct ioband_group *p;
+ unsigned long flags;
+
+ spin_lock_irqsave(&dp->g_lock, flags);
```

```
+ list_for_each_entry(p, &head->c_group_list, c_group_list) {
+  set_group_down(p);
+  if (suspend) {
+   set_group_suspended(p);
+   dprintk(KERN_ERR "ioband suspend: gp(%p)\n", p);
+  }
+
+ }
+ set_group_down(head);
+ if (suspend) {
+  set_group_suspended(head);
+  dprintk(KERN_ERR "ioband suspend: gp(%p)\n", head);
+ }
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+ queue_work(dp->g_ioband_wq, &dp->g_conductor);
+ flush_workqueue(dp->g_ioband_wq);
+}
+
+static void ioband_group_resume_all(struct ioband_group *head)
+{
+ struct ioband_device *dp = head->c_banddev;
+ struct ioband_group *p;
+ unsigned long flags;
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+ list_for_each_entry(p, &head->c_group_list, c_group_list) {
+  clear_group_down(p);
+  clear_group_suspended(p);
+  dprintk(KERN_ERR "ioband resume: gp(%p)\n", p);
+ }
+ clear_group_down(head);
+ clear_group_suspended(head);
+ dprintk(KERN_ERR "ioband resume: gp(%p)\n", head);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+}
+
+/*
+ * Create a new band device:
+ *   parameters:  <device> <device-group-id> <io_throttle> <io_limit>
+ *     <type> <policy> <policy-param...> <group-id:group-param...>
+ */
+static int ioband_ctr(struct dm_target *ti, unsigned int argc, char **argv)
+{
+ struct ioband_group *gp, *defgrp;
+ struct ioband_device *dp;
+ int io_throttle = DEFAULT_IO_THROTTLE;
+ int io_limit = DEFAULT_IO_LIMIT;
+ int i, r = 0;
```

```
+ long val, id;
+ char *param, *endp;
+
+ if (argc < POLICY_PARAM_START) {
+  ti->error = "Requires " __stringify(POLICY_PARAM_START)
+       " or more arguments";
+  return -EINVAL;
+ }
+
+ if (strlen(argv[1]) > IOBAND_NAME_MAX) {
+  ti->error = "Ioband device name is too long";
+  return -EINVAL;
+ }
+ dprintk(KERN_ERR "ioband_ctr ioband device name:%s\n", argv[1]);
+
+ gp = kzalloc(sizeof(struct ioband_group), GFP_KERNEL);
+ if (!gp) {
+  ti->error = "Cannot allocate memory for ioband group";
+  return -ENOMEM;
+ }
+
+ r = dm_get_device(ti, argv[0], 0, ti->len,
+     dm_table_get_mode(ti->table), &gp->c_dev);
+ if (r) {
+  kfree(gp);
+  ti->error = "Device lookup failed";
+  return r;
+ }
+ ti->private = gp;
+
+ val = simple_strtol(argv[2], NULL, 0);
+ if (val > 0)
+  io_throttle = val;
+
+ val = simple_strtol(argv[3], NULL, 0);
+ if (val > 0)
+  io_limit = val;
+ else {
+  struct request_queue *q;
+
+  q = bdev_get_queue(gp->c_dev->bdev);
+  if (!q) {
+   dm_put_device(ti, gp->c_dev);
+   kfree(gp);
+   ti->error = "Can't get queue size";
+   return -ENXIO;
+  }
+  dprintk(KERN_ERR "ioband_ctr nr_requests:%lu\n",
```

```
+       q->nr_requests);
+  io_limit = q->nr_requests;
+ }
+
+ if (io_limit < io_throttle)
+  io_limit = io_throttle;
+ dprintk(KERN_ERR "ioband_ctr io_throttle:%d io_limit:%d\n",
+       io_throttle, io_limit);
+
+ dp = alloc_ioband_device(argv[1], argv[5], io_throttle, io_limit,
+    argc - POLICY_PARAM_START, &argv[POLICY_PARAM_START]);
+ if (!dp) {
+  dm_put_device(ti, gp->c_dev);
+  kfree(gp);
+  ti->error = "Cannot create ioband device";
+  return -EINVAL;
+ }
+
+ for (i = POLICY_PARAM_START, defgrp = NULL; i < argc; i++) {
+  param = strpbrk(argv[i], POLICY_PARAM_DELIM);
+  if (!param)
+   continue;
+  if (param == argv[i]) { /* default ioband group */
+   if (defgrp) {
+    ti->error =
+     "Default ioband group specified twice";
+    r = -EINVAL;
+    goto error;
+   }
+   param++;
+   r = ioband_group_init(gp, NULL, dp,
+       IOBAND_ID_ANY, param);
+   if (r) {
+    ti->error =
+     "Cannot create default ioband group";
+    goto error;
+   }
+   gp->c_target = ti;
+   defgrp = gp;
+   r = ioband_group_type_select(defgrp, argv[4]);
+   if (r) {
+    ti->error = "Cannot set ioband group type";
+    goto error;
+   }
+  } else { /* sub ioband group */
+   if (!defgrp)
+    break;
+   id = simple_strtol(argv[i], &endp, 0);
```

```
+   if (!strchr(POLICY_PARAM_DELIM, *endp)) {
+    ti->error = "Invalid ioband group ID";
+    r = -EINVAL;
+    goto error;
+   }
+   param++;
+   r = ioband_group_attach(defgrp, id, param);
+   if (r) {
+    ti->error = "Cannot create ioband group";
+    goto error;
+   }
+  }
+ }
+
+ if (!defgrp) {
+  ti->error = "No default ioband group specified";
+  r = -EINVAL;
+  goto error;
+ }
+ return 0;
+
+error:
+ dm_put_device(ti, gp->c_dev);
+ if (!defgrp)
+  kfree(gp);
+ else
+  ioband_group_destroy_all(gp);
+ release_ioband_device(dp);
+ return r;
+}
+
+static void ioband_dtr(struct dm_target *ti)
+{
+ struct ioband_group *gp = ti->private;
+ struct ioband_device *dp = gp->c_banddev;
+
+ ioband_group_stop_all(gp, 0);
+ dm_put_device(ti, gp->c_dev);
+ ioband_group_destroy_all(gp);
+ release_ioband_device(dp);
+}
+
+static void ioband_hold_bio(struct ioband_group *gp, struct bio *bio)
+{
+ /* Todo: The list should be split into a read list and a write list */
+ bio_list_add(&gp->c_blocked_bios, bio);
+}
+
```

```
+static struct bio *ioband_pop_bio(struct ioband_group *gp)
+{
+ return bio_list_pop(&gp->c_blocked_bios);
+}
+
+static inline void resume_to_accept_bios(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ if (is_device_blocked(dp)
+     && dp->g_blocked < dp->g_io_limit[0]+dp->g_io_limit[1]) {
+  clear_device_blocked(dp);
+  wake_up_all(&dp->g_waitq);
+ }
+ if (is_group_blocked(gp)) {
+  clear_group_blocked(gp);
+  wake_up_all(&gp->c_waitq);
+ }
+}
+
+static inline int device_should_block(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ if (is_group_down(gp))
+  return 0;
+ if (is_device_blocked(dp))
+  return 1;
+ if (dp->g_blocked >= dp->g_io_limit[0] + dp->g_io_limit[1]) {
+  set_device_blocked(dp);
+  return 1;
+ }
+ return 0;
+}
+
+static inline int group_should_block(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ if (is_group_down(gp))
+  return 0;
+ if (is_group_blocked(gp))
+  return 1;
+ if (dp->g_should_block(gp)) {
+  set_group_blocked(gp);
+  return 1;
+ }
+ return 0;
```

```
+}
+
+static void do_nothing(void) {}
+
+static void prevent_burst_bios(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ if (!current->mm || current->flags & PF_BORROWED_MM) {
+  /*
+   * Kernel threads shouldn't be blocked easily since each of
+   * them may handle BIOs for several groups on several
+   * partitions.
+   */
+  wait_event_lock_irq(dp->g_waitq, !device_should_block(gp),
+      dp->g_lock, do_nothing());
+ } else {
+  wait_event_lock_irq(gp->c_waitq, !group_should_block(gp),
+      dp->g_lock, do_nothing());
+ }
+}
+
+static inline int should_pushback_bio(struct ioband_group *gp)
+{
+ return is_group_suspended(gp) && dm_noflush_suspending(gp->c_target);
+}
+
+static inline void prepare_to_issue(struct ioband_group *gp, struct bio *bio)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ dp->g_prepare_bio(gp, bio);
+ dp->g_issued[bio_data_dir(bio)]++;
+}
+
+static inline int room_for_bio(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ return dp->g_issued[0] < dp->g_io_limit[0]
+  || dp->g_issued[1] < dp->g_io_limit[1] || is_group_down(gp);
+}
+
+static void hold_bio(struct ioband_group *gp, struct bio *bio)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ dp->g_blocked++;
```

```
+ gp->c_blocked++;
+ dp->g_hold_bio(gp, bio);
+}
+
+
+
+static inline int room_for_bio_rw(struct ioband_group *gp, int direct)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ return dp->g_issued[direct] < dp->g_io_limit[direct] ||
+        is_group_down(gp);
+}
+
+static void push_prio_bio(struct ioband_group *gp, struct bio *bio, int direct)
+{
+ if (bio_list_empty(&gp->c_prio_bios))
+   set_prio_queue(gp, direct);
+ bio_list_add(&gp->c_prio_bios, bio);
+ gp->c_prio_blocked++;
+}
+
+static struct bio *pop_prio_bio(struct ioband_group *gp)
+{
+ struct bio *bio = bio_list_pop(&gp->c_prio_bios);
+
+ if (bio_list_empty(&gp->c_prio_bios))
+   clear_prio_queue(gp);
+
+ if (bio)
+   gp->c_prio_blocked--;
+ return bio;
+}
+
+static void release_prio_bios(struct ioband_group *gp,
+    struct bio_list *issue_list, struct bio_list *pushback_list)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct bio *bio;
+ int direct;
+
+ if (bio_list_empty(&gp->c_prio_bios))
+   return;
+ direct = prio_queue_direct(gp);
+ while (room_for_bio_rw(gp, direct)) {
+   bio = pop_prio_bio(gp);
+   if (!bio)
+     return;
```

```
+  dp->g_blocked--;
+  gp->c_blocked--;
+  if (!gp->c_blocked)
+   resume_to_accept_bios(gp);
+  prepare_to_issue(gp, bio);
+  if (is_group_suspended(gp))
+   bio_list_add(pushback_list, bio);
+  else
+   bio_list_add(issue_list, bio);
+ }
+}
+
+static void release_norm_bios(struct ioband_group *gp,
+   struct bio_list *issue_list, struct bio_list *pushback_list)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct bio *bio;
+ int direct;
+
+ while (dp->g_can_submit(gp) && gp->c_blocked - gp->c_prio_blocked) {
+  if (!room_for_bio(gp))
+   return;
+  bio = dp->g_pop_bio(gp);
+  if (!bio)
+   return;
+
+  direct = bio_data_dir(bio);
+  if (!room_for_bio_rw(gp, direct)) {
+   push_prio_bio(gp, bio, direct);
+   continue;
+  }
+  dp->g_blocked--;
+  gp->c_blocked--;
+  if (!gp->c_blocked)
+   resume_to_accept_bios(gp);
+  prepare_to_issue(gp, bio);
+  if (is_group_suspended(gp))
+   bio_list_add(pushback_list, bio);
+  else
+   bio_list_add(issue_list, bio);
+ }
+}
+
+static inline void release_bios(struct ioband_group *gp,
+   struct bio_list *issue_list, struct bio_list *pushback_list)
+{
+
+ release_prio_bios(gp, issue_list, pushback_list);
```

```
+ release_norm_bios(gp, issue_list, pushback_list);
+}
+
+static struct ioband_group *ioband_group_get(
+    struct ioband_group *head, struct bio *bio)
+{
+ struct ioband_group *gp;
+
+ if (!head->c_type->t_getid)
+  return head;
+
+ gp = ioband_group_find(head, head->c_type->t_getid(bio));
+
+ if (!gp)
+  gp = head;
+ return gp;
+}
+
+/*
+ * Start to control the bandwidth once the number of uncompleted BIOs
+ * exceeds the value of "io_throttle".
+ */
+static int ioband_map(struct dm_target *ti, struct bio *bio,
+       union map_info *map_context)
+{
+ struct ioband_group *gp = ti->private;
+ struct ioband_group_stat *bws;
+ struct ioband_device *dp = gp->c_banddev;
+ unsigned long flags;
+
+#if 0 /* not supported yet */
+ if (bio_barrier(bio))
+  return -EOPNOTSUPP;
+#endif
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+ gp = ioband_group_get(gp, bio);
+ prevent_burst_bios(gp);
+ if (should_pushback_bio(gp)) {
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  return DM_MAPIO_REQUEUE;
+ }
+
+ bio->bi_bdev = gp->c_dev->bdev;
+ bio->bi_sector -= ti->begin;
+ bws = &gp->c_stat[bio_data_dir(bio)];
+ bws->sectors += bio_sectors(bio);
+retry:
```

```
+ if (!gp->c_blocked && room_for_bio_rw(gp, bio_data_dir(bio))) {
+  if (dp->g_can_submit(gp)) {
+   prepare_to_issue(gp, bio);
+   bws->immediate++;
+   spin_unlock_irqrestore(&dp->g_lock, flags);
+   return DM_MAPIO_REMAPPED;
+  } else if (dp->g_issued[0] + dp->g_issued[1] == 0) {
+   if (!dp->g_blocked && dp->g_restart_bios(dp)) {
+    dprintk(KERN_ERR "ioband_map: token refilled "
+      "gp:%p bio:%p\n", gp, bio);
+    goto retry;
+   }
+  }
+ }
+ hold_bio(gp, bio);
+ bws->deferred++;
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+
+ return DM_MAPIO_SUBMITTED;
+}
+
+/*
+ * Select the best group to resubmit its BIOs.
+ */
+static struct ioband_group *choose_best_group(struct ioband_device *dp)
+{
+ struct ioband_group *gp;
+ struct ioband_group *best = NULL;
+ int highest = 0;
+ int pri;
+
+ /* Todo: The algorithm should be optimized.
+  *       It would be better to use rbtree.
+  */
+ list_for_each_entry(gp, &dp->g_groups, c_list) {
+  if (!gp->c_blocked || !room_for_bio(gp))
+   continue;
+  if (gp->c_blocked == gp->c_prio_blocked
+   && !room_for_bio_rw(gp, prio_queue_direct(gp))) {
+   continue;
+  }
+  pri = dp->g_can_submit(gp);
+  if (pri > highest) {
+   highest = pri;
+   best = gp;
+  }
+ }
+
```

```
+ return best;
+}
+
+/*
+ * This function is called right after it becomes able to resubmit BIOs.
+ * It selects the best BIOs and passes them to the underlying layer.
+ */
+static void ioband_conduct(struct work_struct *work)
+{
+ struct ioband_device *dp =
+  container_of(work, struct ioband_device, g_conductor);
+ struct ioband_group *gp = NULL;
+ struct bio *bio;
+ unsigned long flags;
+ struct bio_list issue_list, pushback_list;
+
+ bio_list_init(&issue_list);
+ bio_list_init(&pushback_list);
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+retry:
+ while (dp->g_blocked && (gp = choose_best_group(dp)))
+  release_bios(gp, &issue_list, &pushback_list);
+
+ if (dp->g_blocked
+     && dp->g_issued[0] + dp->g_issued[1] < dp->g_io_throttle) {
+  dprintk(KERN_ERR "ioband_conduct: token expired dp:%p "
+   "issued(%d,%d) g_blocked(%d)\n", dp,
+    dp->g_issued[0], dp->g_issued[1], dp->g_blocked);
+  if (dp->g_restart_bios(dp))
+   goto retry;
+ }
+
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+
+ while ((bio = bio_list_pop(&issue_list)))
+  generic_make_request(bio);
+ while ((bio = bio_list_pop(&pushback_list)))
+  bio_endio(bio, -EIO);
+}
+
+static int ioband_end_io(struct dm_target *ti, struct bio *bio,
+      int error, union map_info *map_context)
+{
+ struct ioband_group *gp = ti->private;
+ struct ioband_device *dp = gp->c_banddev;
+ unsigned long flags;
+ int r = error;
```

```
+
+ /*
+  *  XXX: A new error code for device mapper devices should be used
+  *       rather than EIO.
+  */
+ if (error == -EIO && should_pushback_bio(gp)) {
+  /* This ioband device is suspending */
+  r = DM_ENDIO_REQUEUE;
+ }
+ /*
+  * Todo: The algorithm should be optimized to eliminate the spinlock.
+  */
+ spin_lock_irqsave(&dp->g_lock, flags);
+ dp->g_issued[bio_data_dir(bio)]--;
+
+ /*
+  * Todo: It would be better to introduce high/low water marks here
+  *   not to kick the workqueues so often.
+  */
+ if (dp->g_blocked)
+  queue_work(dp->g_ioband_wq, &dp->g_conductor);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+ return r;
+}
+
+static void ioband_presuspend(struct dm_target *ti)
+{
+ struct ioband_group *gp = ti->private;
+ ioband_group_stop_all(gp, 1);
+}
+
+static void ioband_resume(struct dm_target *ti)
+{
+ struct ioband_group *gp = ti->private;
+ ioband_group_resume_all(gp);
+}
+
+
+static void ioband_group_status(struct ioband_group *gp, int *szp,
+     char *result, unsigned int maxlen)
+{
+ struct ioband_group_stat *bws;
+ unsigned long reqs;
+ int i, sz = *szp; /* used in DMEMIT() */
+
+ DMEMIT(" %d", gp->c_id);
+ for (i = 0; i < 2; i++) {
+  bws = &gp->c_stat[i];
```

```
+  reqs = bws->immediate + bws->deferred;
+  DMEMIT(" %lu %lu %lu",
+   bws->immediate + bws->deferred, bws->deferred,
+   bws->sectors);
+ }
+ *szp = sz;
+}
+
+static int ioband_status(struct dm_target *ti, status_type_t type,
+   char *result, unsigned int maxlen)
+{
+ struct ioband_group *gp = ti->private, *p;
+ struct ioband_device *dp = gp->c_banddev;
+ int sz = 0; /* used in DMEMIT() */
+ unsigned long flags;
+
+ switch (type) {
+ case STATUSTYPE_INFO:
+  spin_lock_irqsave(&dp->g_lock, flags);
+  DMEMIT("%s", dp->g_name);
+  ioband_group_status(gp, &sz, result, maxlen);
+  list_for_each_entry(p, &gp->c_group_list, c_group_list)
+   ioband_group_status(p, &sz, result, maxlen);
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  break;
+
+ case STATUSTYPE_TABLE:
+  spin_lock_irqsave(&dp->g_lock, flags);
+  DMEMIT("%s %s %d %d %s %s",
+    gp->c_dev->name, dp->g_name,
+    dp->g_io_throttle, dp->g_io_limit[0],
+    gp->c_type->t_name, dp->g_policy->p_name);
+  dp->g_show(gp, &sz, result, maxlen);
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  break;
+ }
+
+ return 0;
+}
+
+static int ioband_group_type_select(struct ioband_group *gp, char *name)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct group_type *t;
+ unsigned long flags;
+
+ for (t = dm_ioband_group_type; (t->t_name); t++) {
+  if (!strcmp(name, t->t_name))
```

```
+   break;
+ }
+ if (!t->t_name) {
+  DMWARN("ioband type select: %s isn't supported.", name);
+  return -EINVAL;
+ }
+ spin_lock_irqsave(&dp->g_lock, flags);
+ if (!list_empty(&gp->c_group_list)) {
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  return -EBUSY;
+ }
+ gp->c_type = t;
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+
+ return 0;
+}
+
+static int split_string(char *s, char **v)
+{
+ long id = IOBAND_ID_ANY;
+ char *p, *q;
+
+ p = strsep(&s, POLICY_PARAM_DELIM);
+ q = strsep(&s, POLICY_PARAM_DELIM);
+ if (!q) {
+  *v = p;
+ } else {
+  id = simple_strtol(p, NULL, 0);
+  *v = q;
+ }
+ return id;
+}
+
+static int ioband_set_param(struct ioband_group *gp, char *cmd, char *value)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ char *val_str;
+ int id;
+ unsigned long flags;
+ int r;
+
+ id = split_string(value, &val_str);
+
+ spin_lock_irqsave(&dp->g_lock, flags);
+ if (id != IOBAND_ID_ANY) {
+  gp = ioband_group_find(gp, id);
+  if (!gp) {
+   spin_unlock_irqrestore(&dp->g_lock, flags);
```

```
+   DMWARN("ioband_set_param: id=%d not found.", id);
+   return -EINVAL;
+  }
+ }
+ r = dp->g_set_param(gp, cmd, val_str);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+ return r;
+}
+
+static int ioband_group_attach(struct ioband_group *gp, int id, char *param)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct ioband_group *sub_gp;
+ int r;
+
+ if (id < 0) {
+  DMWARN("ioband_group_attach: invalid id:%d", id);
+  return -EINVAL;
+ }
+ if (!gp->c_type->t_getid) {
+  DMWARN("ioband_group_attach: "
+         "no ioband group type is specified");
+  return -EINVAL;
+ }
+
+ sub_gp = kzalloc(sizeof(struct ioband_group), GFP_KERNEL);
+ if (!sub_gp)
+  return -ENOMEM;
+
+ r = ioband_group_init(sub_gp, gp, dp, id, param);
+ if (r < 0) {
+  kfree(sub_gp);
+  return r;
+ }
+ return 0;
+}
+
+static int ioband_group_detach(struct ioband_group *gp, int id)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct ioband_group *sub_gp;
+ unsigned long flags;
+
+ if (id < 0) {
+  DMWARN("ioband_group_detach: invalid id:%d", id);
+  return -EINVAL;
+ }
+ spin_lock_irqsave(&dp->g_lock, flags);
```

```
+ sub_gp = ioband_group_find(gp, id);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+ if (!sub_gp) {
+  DMWARN("ioband_group_detach: invalid id:%d", id);
+  return -EINVAL;
+ }
+ ioband_group_stop(sub_gp);
+ spin_lock_irqsave(&dp->g_lock, flags);
+ ioband_group_release(sub_gp);
+ spin_unlock_irqrestore(&dp->g_lock, flags);
+ return 0;
+}
+
+/*
+ * Message parameters:
+ * "policy"      <name>
+ *       ex)
+ *  "policy" "weight"
+ * "type"        "none"|"pid"|"pgrp"|"node"|"cpuset"|"cgroup"|"user"|"gid"
+ *  "io_throttle" <value>
+ *  "io_limit"    <value>
+ * "attach"      <group id>
+ * "detach"      <group id>
+ * "any-command" <group id>:<value>
+ *       ex)
+ *  "weight" 0:<value>
+ *  "token"  24:<value>
+ */
+static int ioband_message(struct dm_target *ti, unsigned int argc, char **argv)
+{
+ struct ioband_group *gp = ti->private, *p;
+ struct ioband_device *dp = gp->c_banddev;
+ long val = 0;
+ int r = 0;
+ unsigned long flags;
+
+ if (argc == 1 && !strcmp(argv[0], "reset")) {
+  spin_lock_irqsave(&dp->g_lock, flags);
+  memset(gp->c_stat, 0, sizeof(gp->c_stat));
+  list_for_each_entry(p, &gp->c_group_list, c_group_list)
+   memset(p->c_stat, 0, sizeof(p->c_stat));
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  return 0;
+ }
+
+ if (argc != 2) {
+  DMWARN("Unrecognised band message received.");
+  return -EINVAL;
```

```
+ }
+ if (!strcmp(argv[0], "debug")) {
+  ioband_debug = simple_strtol(argv[1], NULL, 0);
+  if (ioband_debug < 0)
+   ioband_debug = 0;
+  return 0;
+ } else if (!strcmp(argv[0], "io_throttle")) {
+  val = simple_strtol(argv[1], NULL, 0);
+  spin_lock_irqsave(&dp->g_lock, flags);
+  if (val < 0 ||
+   val > dp->g_io_limit[0] || val > dp->g_io_limit[1]) {
+   spin_unlock_irqrestore(&dp->g_lock, flags);
+   return -EINVAL;
+  }
+  dp->g_io_throttle = (val == 0) ? DEFAULT_IO_THROTTLE : val;
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  ioband_set_param(gp, argv[0], argv[1]);
+  return 0;
+ } else if (!strcmp(argv[0], "io_limit")) {
+  val = simple_strtol(argv[1], NULL, 0);
+  if (val == 0) {
+   struct request_queue *q;
+
+   q = bdev_get_queue(gp->c_dev->bdev);
+   if (!q)
+    return -ENXIO;
+   val = q->nr_requests;
+  }
+  spin_lock_irqsave(&dp->g_lock, flags);
+  if (val < dp->g_io_throttle) {
+   spin_unlock_irqrestore(&dp->g_lock, flags);
+   return -EINVAL;
+  }
+  dp->g_io_limit[0] = val;
+  dp->g_io_limit[1] = val;
+  spin_unlock_irqrestore(&dp->g_lock, flags);
+  ioband_set_param(gp, argv[0], argv[1]);
+  return 0;
+ } else if (!strcmp(argv[0], "type")) {
+  return ioband_group_type_select(gp, argv[1]);
+ } else if (!strcmp(argv[0], "attach")) {
+  long id = simple_strtol(argv[1], NULL, 0);
+  return ioband_group_attach(gp, id, NULL);
+ } else if (!strcmp(argv[0], "detach")) {
+  long id = simple_strtol(argv[1], NULL, 0);
+  return ioband_group_detach(gp, id);
+ } else {
+  /* message anycommand <group-id>:<value> */
```

```
+  r = ioband_set_param(gp, argv[0], argv[1]);
+  if (r < 0)
+   DMWARN("Unrecognised band message received.");
+  return r;
+ }
+ return 0;
+}
+
+static struct target_type ioband_target = {
+ .name      = "ioband",
+ .module      = THIS_MODULE,
+ .version     = {1, 0, 0},
+ .ctr       = ioband_ctr,
+ .dtr       = ioband_dtr,
+ .map       = ioband_map,
+ .end_io      = ioband_end_io,
+ .presuspend  = ioband_presuspend,
+ .resume      = ioband_resume,
+ .status      = ioband_status,
+ .message     = ioband_message,
+};
+
+static int __init dm_ioband_init(void)
+{
+ int r;
+
+ r = dm_register_target(&ioband_target);
+ if (r < 0) {
+  DMERR("register failed %d", r);
+  return r;
+ }
+ return r;
+}
+
+static void __exit dm_ioband_exit(void)
+{
+ int r;
+
+ r = dm_unregister_target(&ioband_target);
+ if (r < 0)
+  DMERR("unregister failed %d", r);
+}
+
+module_init(dm_ioband_init);
+module_exit(dm_ioband_exit);
+
+MODULE_DESCRIPTION(DM_NAME " I/O bandwidth control");
+MODULE_AUTHOR("Hirokazu Takahashi <taka@valinux.co.jp>, "
```

```
+       "Ryo Tsuruta <ryov@valinux.co.jp");
+MODULE_LICENSE("GPL");
diff -uprN linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband-policy.c
linux-2.6.26-rc2-mm1/drivers/md/dm-ioband-policy.c
--- linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband-policy.c 1970-01-01 09:00:00.000000000
+0900
+++ linux-2.6.26-rc2-mm1/drivers/md/dm-ioband-policy.c 2008-05-19 14:22:37.000000000 +0900
@@ -0,0 +1,302 @@
+/*
+ * Copyright (C) 2008 VA Linux Systems Japan K.K.
+ *
+ *  I/O bandwidth control
+ *
+ * This file is released under the GPL.
+ */
+#include <linux/bio.h>
+#include <linux/workqueue.h>
+#include "dm.h"
+#include "dm-bio-list.h"
+#include "dm-ioband.h"
+
+/*
+ * The following functions determine when and which BIOs should
+ * be submitted to control the I/O flow.
+ * It is possible to add a new BIO scheduling policy with it.
+ */
+
+
+/*
+ * Functions for weight balancing policy based on the number of I/Os.
+ */
+#define DEFAULT_WEIGHT 100
+#define DEFAULT_TOKENBASE 8192
+#define IOBAND_IOPRIO_BASE 100
+
+static int make_global_epoch(struct ioband_device *dp)
+{
+ dp->g_epoch++;
+#if 0 /* this will also work correctly */
+ if (dp->g_blocked)
+  queue_work(dp->g_ioband_wq, &dp->g_conductor);
+ return 0;
+#endif
+ dprintk(KERN_ERR "make_epoch %d --> %d\n",
+       dp->g_epoch-1, dp->g_epoch);
+ return 1;
+}
+
```

```
+static inline int make_epoch(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+
+ if (gp->c_my_epoch != dp->g_epoch) {
+  gp->c_my_epoch = dp->g_epoch;
+  return 1;
+ }
+ return 0;
+}
+
+static inline int iopriority(struct ioband_group *gp)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ int iopri;
+
+ iopri = gp->c_token*IOBAND_IOPRIO_BASE/gp->c_token_init_value + 1;
+ if (gp->c_my_epoch != dp->g_epoch)
+  iopri += IOBAND_IOPRIO_BASE;
+ if (is_group_down(gp))
+  iopri += IOBAND_IOPRIO_BASE*2;
+
+ return iopri;
+}
+
+static int is_token_left(struct ioband_group *gp)
+{
+ if (gp->c_token > 0)
+  return iopriority(gp);
+
+ if (make_epoch(gp) || is_group_down(gp)) {
+  gp->c_token = gp->c_token_init_value;
+  dprintk(KERN_ERR "refill token: gp:%p token:%d\n",
+       gp, gp->c_token);
+  return iopriority(gp);
+ }
+ return 0;
+}
+
+static void prepare_token(struct ioband_group *gp, struct bio *bio)
+{
+ gp->c_token--;
+}
+
+static void set_weight(struct ioband_group *gp, int new)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ struct ioband_group *p;
```

```
+
+ dp->g_weight_total += (new - gp->c_weight);
+ gp->c_weight = new;
+
+ list_for_each_entry(p, &dp->g_groups, c_list) {
+  /* Fixme: it might overflow */
+  p->c_token = p->c_token_init_value =
+     dp->g_token_base * p->c_weight / dp->g_weight_total + 1;
+  p->c_limit =
+     (dp->g_io_limit[0] + dp->g_io_limit[1]) *
+     p->c_weight / dp->g_weight_total + 1;
+ }
+}
+
+static int policy_weight_ctr(struct ioband_group *gp, char *arg)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ char *endp;
+ long weight;
+
+ if (!arg)
+  weight = DEFAULT_WEIGHT;
+ else {
+  weight = simple_strtol(arg, &endp, 0);
+  if (*endp != '\0' || weight <= 0)
+   return -EINVAL;
+ }
+
+ gp->c_my_epoch = dp->g_epoch;
+ gp->c_weight = 0;
+ set_weight(gp, weight);
+ return 0;
+}
+
+static void policy_weight_dtr(struct ioband_group *gp)
+{
+ set_weight(gp, 0);
+}
+
+static int policy_weight_param(struct ioband_group *gp, char *cmd, char *value)
+{
+ struct ioband_device *dp = gp->c_banddev;
+ long val = simple_strtol(value, NULL, 0);
+ int r = 0;
+
+ if (!strcmp(cmd, "weight")) {
+  if (val > 0)
+   set_weight(gp, val);
```

```
+  else
+   r = -EINVAL;
+ } else if (!strcmp(cmd, "token")) {
+  if (val > 0) {
+   dp->g_token_base = val;
+   set_weight(gp, gp->c_weight);
+  } else
+   r = -EINVAL;
+ } else if (!strcmp(cmd, "io_limit")) {
+  set_weight(gp, gp->c_weight);
+ } else {
+  r = -EINVAL;
+ }
+ return r;
+}
+
+static void policy_weight_show(struct ioband_group *gp, int *szp,
+     char *result, unsigned int maxlen)
+{
+ struct ioband_group *p;
+ struct ioband_device *dp = gp->c_banddev;
+ int sz = *szp; /* used in DMEMIT() */
+
+ DMEMIT(" %d :%d", dp->g_token_base, gp->c_weight);
+ list_for_each_entry(p, &gp->c_group_list, c_group_list)
+  DMEMIT(" %d:%d", p->c_id, p->c_weight);
+ *szp = sz;
+}
+
+static int is_queue_full(struct ioband_group *gp)
+{
+ return gp->c_blocked >= gp->c_limit;
+}
+
+
+/*
+ *  <Method>      <description>
+ * g_can_submit   : To determine whether a given group has the right to
+ *                  submit BIOs. The larger the return value the higher the
+ *                  priority to submit. Zero means it has no right.
+ * g_prepare_bio  : Called right before submitting each BIO.
+ * g_restart_bios : Called if this ioband device has some BIOs blocked but none
+ *                  of them can be submitted now. This method has to
+ *                  reinitialize the data to restart to submit BIOs and return
+ *                  0 or 1.
+ *                  The return value 0 means that it has become able to submit
+ *                  them now so that this ioband device will continue its work.
+ *                  The return value 1 means that it is still unable to submit
```

```
+ *               them so that this device will stop its work. And this
+ *               policy module has to reactivate the device when it gets
+ *               to be able to submit BIOs.
+ * g_hold_bio     : To hold a given BIO until it is submitted.
+ *               The default function is used when this method is undefined.
+ * g_pop_bio      : To select and get the best BIO to submit.
+ * g_group_ctr    : To initalize the policy own members of struct ioband_group.
+ * g_group_dtr    : Called when struct ioband_group is removed.
+ * g_set_param    : To update the policy own date.
+ *               The parameters can be passed through "dmsetup message"
+ *               command.
+ * g_should_block : Called every time this ioband device receive a BIO.
+ *               Return 1 if a given group can't receive any more BIOs,
+ *               otherwise return 0.
+ * g_show         : Show the configuration.
+ */
+static int policy_weight_init(struct ioband_device *dp, int argc, char **argv)
+{
+ char *endp;
+ long val;
+
+ if (argc < 1)
+  return -EINVAL;
+
+ val = simple_strtol(argv[0], &endp, 0);
+ if (*endp != '\0' || val < 0)
+  return -EINVAL;
+ if (val == 0)
+  val = DEFAULT_TOKENBASE;
+
+ dp->g_can_submit = is_token_left;
+ dp->g_prepare_bio = prepare_token;
+ dp->g_restart_bios = make_global_epoch;
+ dp->g_group_ctr = policy_weight_ctr;
+ dp->g_group_dtr = policy_weight_dtr;
+ dp->g_set_param = policy_weight_param;
+ dp->g_should_block = is_queue_full;
+ dp->g_show  = policy_weight_show;
+
+ dp->g_token_base = val;
+ dp->g_epoch = 0;
+ dp->g_weight_total = 0;
+ return 0;
+}
+/* weight balancing policy based on the number of I/Os. --- End --- */
+
+
+/*
```

```
+ * Functions for weight balancing policy based on I/O size.
+ * It just borrows a lot of functions from the regular weight balancing policy.
+ */
+
+static int w2_is_token_left(struct ioband_group *gp)
+{
+ if (gp->c_token > 0)
+  return iopriority(gp);
+
+ if (is_group_down(gp)) {
+  gp->c_token = gp->c_token_init_value;
+  return iopriority(gp);
+ }
+ if (make_epoch(gp)) {
+  gp->c_token += gp->c_token_init_value;
+  dprintk(KERN_ERR "refill token: gp:%p token:%d->%d\n",
+   gp, gp->c_token - gp->c_token_init_value, gp->c_token);
+  if (gp->c_token > 0)
+   return iopriority(gp);
+  dprintk(KERN_ERR "refill token: yet empty gp:%p token:%d\n",
+       gp, gp->c_token);
+ }
+ return 0;
+}
+
+static void w2_prepare_token(struct ioband_group *gp, struct bio *bio)
+{
+ gp->c_token -= bio_sectors(bio);
+}
+
+static int w2_policy_weight_init(struct ioband_device *dp,
+       int argc, char **argv)
+{
+ char *endp;
+ long val;
+ int r = 0;
+
+ if (argc < 1)
+  return -EINVAL;
+
+ val = simple_strtol(argv[0], &endp, 0);
+ if (*endp != '\0' || val < 0)
+  return -EINVAL;
+ if (val == 0)
+  val = DEFAULT_TOKENBASE << (PAGE_SHIFT - 9);
+
+ r = policy_weight_init(dp, argc, argv);
+ if (r < 0)
```

```
+  return r;
+
+ dp->g_can_submit = w2_is_token_left;
+ dp->g_prepare_bio = w2_prepare_token;
+ dp->g_token_base = val;
+ return 0;
+}
+/* weight balancing policy based on I/O size. --- End --- */
+
+
+static int policy_default_init(struct ioband_device *dp,
+     int argc, char **argv) /*XXX*/
+{
+ return policy_weight_init(dp, argc, argv); /* temp */
+}
+
+struct policy_type dm_ioband_policy_type[] = {
+ {"default", policy_default_init},
+ {"weight", policy_weight_init},
+ {"weight-iosize", w2_policy_weight_init},
+ {NULL,     policy_default_init}
+};
diff -uprN linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband-type.c
linux-2.6.26-rc2-mm1/drivers/md/dm-ioband-type.c
--- linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband-type.c 1970-01-01 09:00:00.000000000
+0900
+++ linux-2.6.26-rc2-mm1/drivers/md/dm-ioband-type.c 2008-05-19 14:22:37.000000000 +0900
@@ -0,0 +1,76 @@
+/*
+ * Copyright (C) 2008 VA Linux Systems Japan K.K.
+ *
+ *  I/O bandwidth control
+ *
+ * This file is released under the GPL.
+ */
+#include <linux/bio.h>
+#include "dm.h"
+#include "dm-bio-list.h"
+#include "dm-ioband.h"
+
+/*
+ * Any I/O bandwidth can be divided into several bandwidth groups, each of which
+ * has its own unique ID. The following functions are called to determine
+ * which group a given BIO belongs to and return the ID of the group.
+ */
+
+/* ToDo: unsigned long value would be better for group ID */
+
```

```
+static int ioband_process_id(struct bio *bio)
+{
+ /*
+  * This function will work for KVM and Xen.
+  */
+ return (int)current->tgid;
+}
+
+static int ioband_process_group(struct bio *bio)
+{
+ return (int)task_pgrp_nr(current);
+}
+
+static int ioband_uid(struct bio *bio)
+{
+ return (int)current->uid;
+}
+
+static int ioband_gid(struct bio *bio)
+{
+ return (int)current->gid;
+}
+
+static int ioband_cpuset(struct bio *bio)
+{
+ return 0; /* not implemented yet */
+}
+
+static int ioband_node(struct bio *bio)
+{
+ return 0; /* not implemented yet */
+}
+
+static int ioband_cgroup(struct bio *bio)
+{
+  /*
+   * This function should return the ID of the cgroup which issued "bio".
+   * The ID of the cgroup which the current process belongs to won't be
+   * suitable ID for this purpose, since some BIOs will be handled by kernel
+   * threads like aio or pdflush on behalf of the process requesting the BIOs.
+   */
+ return 0; /* not implemented yet */
+}
+
+struct group_type dm_ioband_group_type[] = {
+ {"none",   NULL},
+ {"pgrp",   ioband_process_group},
+ {"pid",    ioband_process_id},
```

```
+ {"node",   ioband_node},
+ {"cpuset", ioband_cpuset},
+ {"cgroup", ioband_cgroup},
+ {"user",   ioband_uid},
+ {"uid",    ioband_uid},
+ {"gid",    ioband_gid},
+ {NULL,     NULL}
+};
diff -uprN linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband.h
linux-2.6.26-rc2-mm1/drivers/md/dm-ioband.h
--- linux-2.6.26-rc2-mm1.orig/drivers/md/dm-ioband.h 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.26-rc2-mm1/drivers/md/dm-ioband.h 2008-05-19 14:22:37.000000000 +0900
@@ -0,0 +1,151 @@
+/*
+ * Copyright (C) 2008 VA Linux Systems Japan K.K.
+ *
+ *  I/O bandwidth control
+ *
+ * This file is released under the GPL.
+ */
+
+#include <linux/version.h>
+#include <linux/wait.h>
+
+#define DEFAULT_IO_THROTTLE 4
+#define DEFAULT_IO_LIMIT 128
+#define IOBAND_NAME_MAX 31
+#define IOBAND_ID_ANY (-1)
+
+struct ioband_group;
+
+struct ioband_device {
+ struct list_head g_groups;
+ struct work_struct g_conductor;
+ struct workqueue_struct *g_ioband_wq;
+ int g_io_throttle;
+ int g_io_limit[2];
+ int g_issued[2];
+ int g_blocked;
+ spinlock_t g_lock;
+ wait_queue_head_t g_waitq;
+
+ int g_ref;  /* just for debugging */
+ struct list_head g_list;
+ int g_flags;
+ char g_name[IOBAND_NAME_MAX + 1];
+ struct policy_type *g_policy;
+
```

```
+ /* policy dependent */
+ int (*g_can_submit)(struct ioband_group *);
+ void (*g_prepare_bio)(struct ioband_group *, struct bio *);
+ int (*g_restart_bios)(struct ioband_device *);
+ void (*g_hold_bio)(struct ioband_group *, struct bio *);
+ struct bio * (*g_pop_bio)(struct ioband_group *);
+ int (*g_group_ctr)(struct ioband_group *, char *);
+ void (*g_group_dtr)(struct ioband_group *);
+ int (*g_set_param)(struct ioband_group *, char *cmd, char *value);
+ int (*g_should_block)(struct ioband_group *);
+ void (*g_show)(struct ioband_group *, int *, char *, unsigned int);
+
+ /* members for weight balancing policy */
+ int g_epoch;
+ int g_weight_total;
+ int g_token_base;
+
+};
+
+struct ioband_group_stat {
+ unsigned long sectors;
+ unsigned long immediate;
+ unsigned long deferred;
+};
+
+struct ioband_group {
+ struct list_head c_list;
+ struct ioband_device *c_banddev;
+ struct dm_dev *c_dev;
+ struct dm_target *c_target;
+ struct bio_list c_blocked_bios;
+ struct bio_list c_prio_bios;
+ struct list_head c_group_list;
+ int c_id; /* should be unsigned long or unsigned long long */
+ char c_name[IOBAND_NAME_MAX + 1]; /* rfu */
+ int c_blocked;
+ int c_prio_blocked;
+ wait_queue_head_t c_waitq;
+ int c_flags;
+ struct ioband_group_stat c_stat[2]; /* hold rd/wr status */
+ struct group_type *c_type;
+
+ /* members for weight balancing policy */
+ int c_weight;
+ int c_my_epoch;
+ int c_token;
+ int c_token_init_value;
+ int c_limit;
```

```
+
+ /* rfu */
+ /* struct bio_list c_ordered_tag_bios; */
+};
+
+#define DEV_BIO_BLOCKED 1
+
+#define set_device_blocked(dp)  ((dp)->g_flags |= DEV_BIO_BLOCKED)
+#define clear_device_blocked(dp) ((dp)->g_flags &= ~DEV_BIO_BLOCKED)
+#define is_device_blocked(dp)  ((dp)->g_flags & DEV_BIO_BLOCKED)
+
+
+#define IOG_PRIO_BIO_WRITE 1
+#define IOG_PRIO_QUEUE  2
+#define IOG_BIO_BLOCKED  4
+#define IOG_GOING_DOWN  8
+#define IOG_SUSPENDED  16
+
+#define set_group_blocked(gp)  ((gp)->c_flags |= IOG_BIO_BLOCKED)
+#define clear_group_blocked(gp)  ((gp)->c_flags &= ~IOG_BIO_BLOCKED)
+#define is_group_blocked(gp)  ((gp)->c_flags & IOG_BIO_BLOCKED)
+
+#define set_group_down(gp)  ((gp)->c_flags |= IOG_GOING_DOWN)
+#define clear_group_down(gp)  ((gp)->c_flags &= ~IOG_GOING_DOWN)
+#define is_group_down(gp)  ((gp)->c_flags & IOG_GOING_DOWN)
+
+#define set_group_suspended(gp)  ((gp)->c_flags |= IOG_SUSPENDED)
+#define clear_group_suspended(gp) ((gp)->c_flags &= ~IOG_SUSPENDED)
+#define is_group_suspended(gp)  ((gp)->c_flags & IOG_SUSPENDED)
+
+#define set_prio_read(gp)  ((gp)->c_flags |= IOG_PRIO_QUEUE)
+#define clear_prio_read(gp)  ((gp)->c_flags &= ~IOG_PRIO_QUEUE)
+#define is_prio_read(gp) \
+ ((gp)->c_flags & (IOG_PRIO_QUEUE|IOG_PRIO_BIO_WRITE) == IOG_PRIO_QUEUE)
+
+#define set_prio_write(gp) \
+ ((gp)->c_flags |= (IOG_PRIO_QUEUE|IOG_PRIO_BIO_WRITE))
+#define clear_prio_write(gp) \
+ ((gp)->c_flags &= ~(IOG_PRIO_QUEUE|IOG_PRIO_BIO_WRITE))
+#define is_prio_write(gp) \
+ ((gp)->c_flags & (IOG_PRIO_QUEUE|IOG_PRIO_BIO_WRITE) == \
+  (IOG_PRIO_QUEUE|IOG_PRIO_BIO_WRITE))
+
+#define set_prio_queue(gp, direct) \
+ ((gp)->c_flags |= (IOG_PRIO_QUEUE|direct))
+#define clear_prio_queue(gp)  clear_prio_write(gp)
+#define is_prio_queue(gp)  ((gp)->c_flags & IOG_PRIO_QUEUE)
+#define prio_queue_direct(gp)  ((gp)->c_flags & IOG_PRIO_BIO_WRITE)
```

```
+
+
+struct policy_type {
+ const char *p_name;
+ int   (*p_policy_init)(struct ioband_device *, int, char **);
+};
+
+extern struct policy_type dm_ioband_policy_type[];
+
+struct group_type {
+ const char *t_name;
+ int   (*t_getid)(struct bio *);
+};
+
+extern struct group_type dm_ioband_group_type[];
+
+/* Just for debugging */
+extern long ioband_debug;
+#define dprintk(format, a...) \
+ if (ioband_debug > 0) ioband_debug--, printk(format, ##a)
```

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

---

Subject: [PATCH 2/2] dm-ioband: I/O bandwidth controller v1.0.0: Document
Posted by Ryo Tsuruta on Mon, 19 May 2008 08:24:28 GMT
View Forum Message <> Reply to Message

Here is the document of dm-ioband.

Based on 2.6.26-rc2-mm1
Signed-off-by: Ryo Tsuruta <ryov@valinux.co.jp>
Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

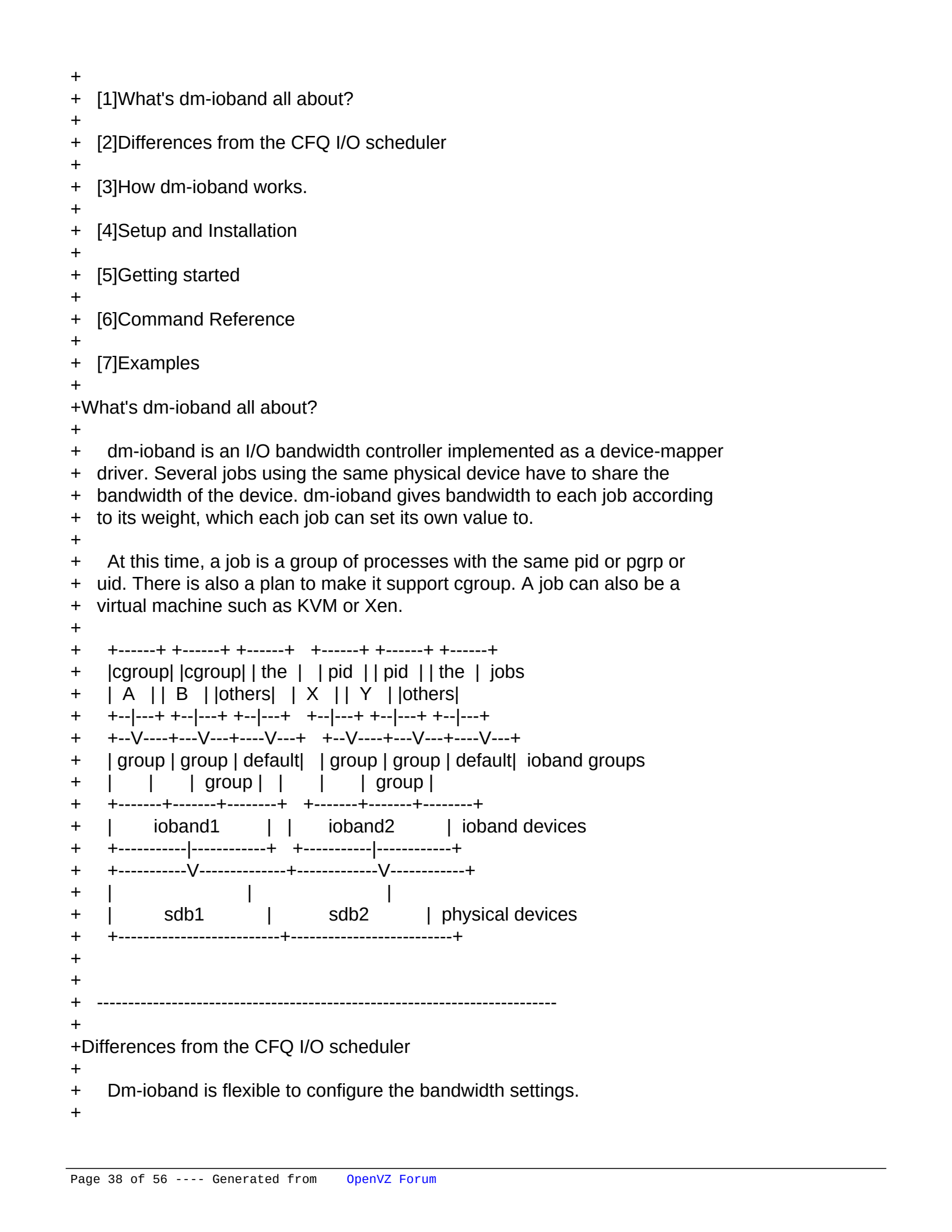
```
diff -uprN linux-2.6.26-rc2-mm1.orig/Documentation/device-mapper/ioband.txt
linux-2.6.26-rc2-mm1/Documentation/device-mapper/ioband.txt
--- linux-2.6.26-rc2-mm1.orig/Documentation/device-mapper/ioband.txt 1970-01-01
09:00:00.000000000 +0900
+++ linux-2.6.26-rc2-mm1/Documentation/device-mapper/ioband.txt 2008-05-19
14:22:37.000000000 +0900
@@ -0,0 +1,899 @@
+                  Block I/O bandwidth control: dm-ioband
+
+          ------------------------------------------------------
+
+   Table of Contents
```

+
+ [1]What's dm-ioband all about?
+
+ [2]Differences from the CFQ I/O scheduler
+
+ [3]How dm-ioband works.
+
+ [4]Setup and Installation
+
+ [5]Getting started
+
+ [6]Command Reference
+
+ [7]Examples
+

+What's dm-ioband all about?
+

+ dm-ioband is an I/O bandwidth controller implemented as a device-mapper
+ driver. Several jobs using the same physical device have to share the
+ bandwidth of the device. dm-ioband gives bandwidth to each job according
+ to its weight, which each job can set its own value to.
+

+ At this time, a job is a group of processes with the same pid or pgrp or
+ uid. There is also a plan to make it support cgroup. A job can also be a
+ virtual machine such as KVM or Xen.
+



+
+
+ ---

+Differences from the CFQ I/O scheduler
+

+ Dm-ioband is flexible to configure the bandwidth settings.
+

+   Dm-ioband can work with any type of I/O scheduler such as the NOOP
+ scheduler, which is often chosen for high-end storages, since it is
+ implemented outside the I/O scheduling layer. It allows both of partition
+ based bandwidth control and job --- a group of processes --- based
+ control. In addition, it can set different configuration on each physical
+ device to control its bandwidth.
+
+   Meanwhile the current implementation of the CFQ scheduler has 8 IO
+ priority levels and all jobs whose processes have the same IO priority
+ share the bandwidth assigned to this level between them. And IO priority
+ is an attribute of a process so that it equally effects to all block
+ devices.
+
+ --
+
+How dm-ioband works.
+
+   Every ioband device has one ioband group, which by default is called the
+ default group.
+
+   Ioband devices can also have extra ioband groups in them. Each ioband
+ group has a job to support and a weight. Proportional to the weight,
+ dm-ioband gives tokens to the group.
+
+   A group passes on I/O requests that its job issues to the underlying
+ layer so long as it has tokens left, while requests are blocked if there
+ aren't any tokens left in the group. Tokens are refilled once all of
+ groups that have requests on a given physical device use up their tokens.
+
+   There are two policies for token consumption. One is that a token is
+ consumed for each I/O request. The other is that a token is consumed for
+ each I/O sector, for example, one I/O request which consists of
+ 4Kbytes(512bytes * 8 sectors) read consumes 8 tokens. A user can choose
+ either policy.
+
+   With this approach, a job running on an ioband group with large weight
+ is guaranteed a wide I/O bandwidth.
+
+ --
+
+Setup and Installation
+
+   Build a kernel with these options enabled:
+
+   CONFIG_MD
+   CONFIG_BLK_DEV_DM
+   CONFIG_DM_IOBAND
+

+
+   If compiled as module, use modprobe to load dm-ioband.
+
+   # make modules
+   # make modules_install
+   # depmod -a
+   # modprobe dm-ioband
+
+
+   "dmsetup targets" command shows all available device-mapper targets.
+   "ioband" is displayed if dm-ioband has been loaded.
+
+   # dmsetup targets
+   ioband   v1.0.0
+
+
+   --
+
+Getting started
+
+   The following is a brief description how to control the I/O bandwidth of
+   disks. In this description, we'll take one disk with two partitions as an
+   example target.
+
+   --
+
+ Create and map ioband devices
+
+   Create two ioband devices "ioband1" and "ioband2". "ioband1" is mapped
+   to "/dev/sda1" and has a weight of 40. "ioband2" is mapped to "/dev/sda2"
+   and has a weight of 10. "ioband1" can use 80% --- 40/(40+10)*100 --- of
+   the bandwidth of the physical disk "/dev/sda" while "ioband2" can use 20%.
+
+   # echo "0 $(blockdev --getsize /dev/sda1) ioband /dev/sda1 1 0 0 none" \
+   "weight 0 :40" | dmsetup create ioband1
+   # echo "0 $(blockdev --getsize /dev/sda2) ioband /dev/sda2 1 0 0 none" \
+   "weight 0 :10" | dmsetup create ioband2
+
+
+   If the commands are successful then the device files
+   "/dev/mapper/ioband1" and "/dev/mapper/ioband2" will have been created.
+
+   --
+
+ Additional bandwidth control
+
+   In this example two extra ioband groups are created on "ioband1". The
+   first group consists of all the processes with user-id 1000 and the second

+ group consists of all the processes with user-id 2000. Their weights are
+ 30 and 20 respectively.
+
+ # dmsetup message ioband1 0 type user
+ # dmsetup message ioband1 0 attach 1000
+ # dmsetup message ioband1 0 attach 2000
+ # dmsetup message ioband1 0 weight 1000:30
+ # dmsetup message ioband1 0 weight 2000:20
+
+
+   Now the processes in the user-id 1000 group can use 30% ---
+ 30/(30+20+40+10)*100 --- of the bandwidth of the physical disk.
+
+ Table 1. Weight assignments
+
+ +---+
+ | ioband device | ioband group | ioband weight |
+ |---------------+--------------------------------+---------------|
+ | ioband1 | user id 1000 | 30 |
+ |---------------+--------------------------------+---------------|
+ | ioband1 | user id 2000 | 20 |
+ |---------------+--------------------------------+---------------|
+ | ioband1 | default group(the other users) | 40 |
+ |---------------+--------------------------------+---------------|
+ | ioband2 | default group | 10 |
+ +---+
+
+ ---
+
+ Remove the ioband devices
+
+   Remove the ioband devices when no longer used.
+
+   # dmsetup remove ioband1
+   # dmsetup remove ioband2
+
+
+ ---
+
+Command Reference
+
+ Create an ioband device
+
+ SYNOPSIS
+
+   dmsetup create IOBAND_DEVICE
+
+ DESCRIPTION

+
+ Create an ioband device with the given name IOBAND_DEVICE.
+ Generally, dmsetup reads a table from standard input. Each line of
+ the table specifies a single target and is of the form:
+
+ start_sector num_sectors "ioband" device_file ioband_device_id \
+ io_throttle io_limit ioband_group_type policy token_base \
+ :weight [ioband_group_id:weight...]
+
+
+ start_sector, num_sectors
+
+ The sector range of the underlying device where
+ dm-ioband maps.
+
+ ioband
+
+ Specify the string "ioband" as a target type.
+
+ device_file
+
+ Underlying device name.
+
+ ioband_device_id
+
+ The ID number for an ioband device. The same ID
+ must be set among the ioband devices that share the
+ same bandwidth, which means they work on the same
+ physical disk.
+
+ io_throttle
+
+ Dm-ioband starts to control the bandwidth when the
+ number of BIOs in progress exceeds this value. If 0
+ is specified, dm-ioband uses the default value.
+
+ io_limit
+
+ Dm-ioband blocks all I/O requests for the
+ IOBAND_DEVICE when the number of BIOs in progress
+ exceeds this value. If 0 is specified, dm-ioband uses
+ the default value.
+
+ ioband_group_type
+
+ Specify how to evaluate the ioband group ID. The
+ type must be one of "none", "user", "gid", "pid" or
+ "pgrp." Specify "none" if you don't need any ioband

+ groups other than the default ioband group.
+
+ policy
+
+ Specify bandwidth control policy. A user can choose either policy "weight" or "weight-iosize."
+
+ weight
+
+ This policy controls bandwidth according to the proportional to the weight of each ioband group based on the number of I/O requests.
+
+ weight-iosize
+
+ This policy controls bandwidth according to the proportional to the weight of each ioband group based on the number of I/O sectors.
+
+ token_base
+
+ The number of tokens which specified by token_base will be distributed to all ioband groups according to the proportional to the weight of each ioband group. If 0 is specified, dm-ioband uses the default value.
+
+ ioband_group_id:weight
+
+ Set the weight of the ioband group specified by ioband_group_id. If ioband_group_id is omitted, the weight is assigned to the default ioband group.
+
+ EXAMPLE
+
+ Create an ioband device with the following parameters:
+
+ * Starting sector = "0"
+
+ * The number of sectors = "$(blockdev --getsize /dev/sda1)"
+
+ * Target type = "ioband"
+
+ * Underlying device name = "/dev/sda1"
+
+ * Ioband device ID = "128"
+

+ * I/O throttle = "10"
+
+ * I/O limit = "400"
+
+ * Ioband group type = "user"
+
+ * Bandwidth control policy = "weight"
+
+ * Token base = "2048"
+
+ * Weight for the default ioband group = "100"
+
+ * Weight for the ioband group 1000 = "80"
+
+ * Weight for the ioband group 2000 = "20"
+
+ * Ioband device name = "ioband1"
+
+ # echo "0 $(blockdev --getsize /dev/sda1) ioband" \
+ "/dev/sda1 128 10 400 user weight 2048 :100 1000:80 2000:20" \
+ | dmsetup create ioband1
+
+
+ Create two device groups (ID=1,2). The bandwidths of these
+ device groups will be individually controlled.
+
+ # echo "0 $(blockdev --getsize /dev/sda1) ioband /dev/sda1 1" \
+ "0 0 none weight 0 :80" | dmsetup create ioband1
+ # echo "0 $(blockdev --getsize /dev/sda2) ioband /dev/sda2 1" \
+ "0 0 none weight 0 :20" | dmsetup create ioband2
+ # echo "0 $(blockdev --getsize /dev/sdb3) ioband /dev/sdb3 2" \
+ "0 0 none weight 0 :60" | dmsetup create ioband3
+ # echo "0 $(blockdev --getsize /dev/sdb4) ioband /dev/sdb4 2" \
+ "0 0 none weight 0 :40" | dmsetup create ioband4
+
+
+ ---
+
+ Remove the ioband device
+
+ SYNOPSIS
+
+ dmsetup remove IOBAND_DEVICE
+
+ DESCRIPTION
+
+ Remove the specified ioband device IOBAND_DEVICE. All the band
+ groups attached to the ioband device are also removed

+ automatically.
+
+ EXAMPLE
+
+ Remove ioband device "ioband1."
+
+ # dmsetup remove ioband1
+
+
+ ---
+
+ Set an ioband group type
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 type TYPE
+
+ DESCRIPTION
+
+ Set the ioband group type of the specified ioband device
+ IOBAND_DEVICE. TYPE must be one of "none", "user", "gid", "pid" or
+ "pgrp." Once the type is set, new ioband groups can be created on
+ IOBAND_DEVICE.
+
+ EXAMPLE
+
+ Set the ioband group type of ioband device "ioband1" to "user."
+
+ # dmsetup message ioband1 0 type user
+
+
+ ---
+
+ Create an ioband group
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 attach ID
+
+ DESCRIPTION
+
+ Create an ioband group and attach it to IOBAND_DEVICE. ID
+ specifies user-id, group-id, process-id or process-group-id
+ depending the ioband group type of IOBAND_DEVICE.
+
+ EXAMPLE
+
+ Create an ioband group which consists of all processes with

+ user-id 1000 and attach it to ioband device "ioband1."
+
+ # dmsetup message ioband1 0 type user
+ # dmsetup message ioband1 0 attach 1000
+
+
+ ---
+
+ Detach the ioband group
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 detach ID
+
+ DESCRIPTION
+
+ Detach the ioband group specified by ID from ioband device
+ IOBAND_DEVICE.
+
+ EXAMPLE
+
+ Detach the ioband group with ID "2000" from ioband device
+ "ioband2."
+
+ # dmsetup message ioband2 0 detach 1000
+
+
+ ---
+
+ Set the weight of an ioband group
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 weight VAL
+
+ dmsetup message IOBAND_DEVICE 0 weight ID:VAL
+
+ DESCRIPTION
+
+ Set the weight of the ioband group specified by ID. Set the
+ weight of the default ioband group of IOBAND_DEVICE if ID isn't
+ specified.
+
+ The following example means that "ioband1" can use 80% ---
+ 40/(40+10)*100 --- of the bandwidth of the physical disk while
+ "ioband2" can use 20%.
+
+ # dmsetup message ioband1 0 weight 40

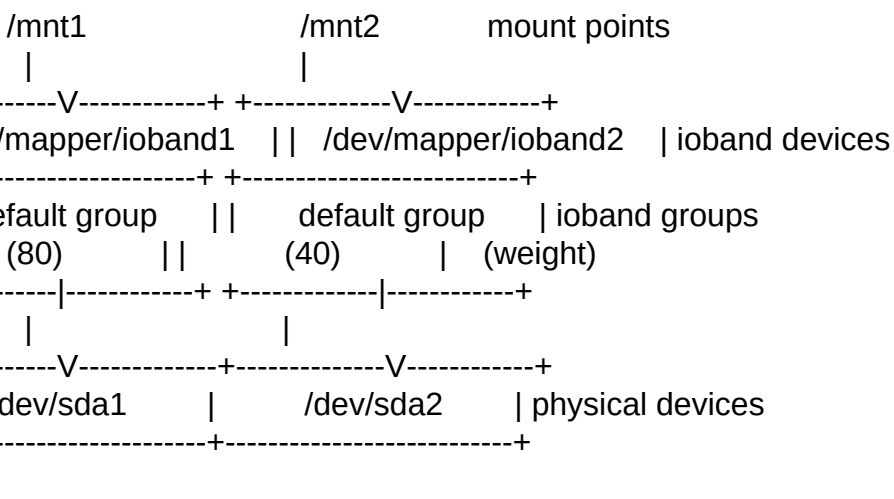
+ # dmsetup message ioband2 0 weight 10
+
+
+ The following lines have the same effect as the above:
+
+ # dmsetup message ioband1 0 weight 4
+ # dmsetup message ioband2 0 weight 1
+
+
+ VAL must be an integer larger than 0. The default value, which
+ is assigned to newly created ioband groups, is 100.
+
+ EXAMPLE
+
+ Set the weight of the default ioband group of "ioband1" to 40.
+
+ # dmsetup message ioband1 0 weight 40
+
+
+ Set the weight of the ioband group of "ioband1" with ID "1000"
+ to 10.
+
+ # dmsetup message ioband1 0 weight 1000:10
+
+
+ ---
+
+ Set the number of tokens
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 token VAL
+
+ DESCRIPTION
+
+ Set the number of tokens to VAL. According to their weight, this
+ number of tokens will be distributed to all the ioband groups on
+ the physical device to which ioband device IOBAND_DEVICE belongs
+ when they use up their tokens.
+
+ VAL must be an integer greater than 0. The default is 2048.
+
+ EXAMPLE
+
+ Set the number of tokens of the physical device to which
+ "ioband1" belongs to 256.
+
+ # dmsetup message ioband1 0 token 256

+
+
+ ---
+
+ Set I/O throttling
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 io_throttle VAL
+
+ DESCRIPTION
+
+ Set the I/O throttling value of the physical disk to which
+ ioband device IOBAND_DEVICE belongs to VAL. Dm-ioband start to
+ control the bandwidth when the number of BIOs in progress on the
+ physical disk exceeds this value.
+
+ EXAMPLE
+
+ Set the I/O throttling value of "ioband1" to 16.
+
+ # dmsetup message ioband1 0 io_throttle 16
+
+
+ ---
+
+ Set I/O limiting
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 io_limit VAL
+
+ DESCRIPTION
+
+ Set the I/O limiting value of the physical disk to which ioband
+ device IOBAND_DEVICE belongs to VAL. Dm-ioband will block all I/O
+ requests for the physical device if the number of BIOs in progress
+ on the physical disk exceeds this value.
+
+ EXAMPLE
+
+ Set the I/O limiting value of "ioband1" to 128.
+
+ # dmsetup message ioband1 0 io_limit 128
+
+
+ ---
+

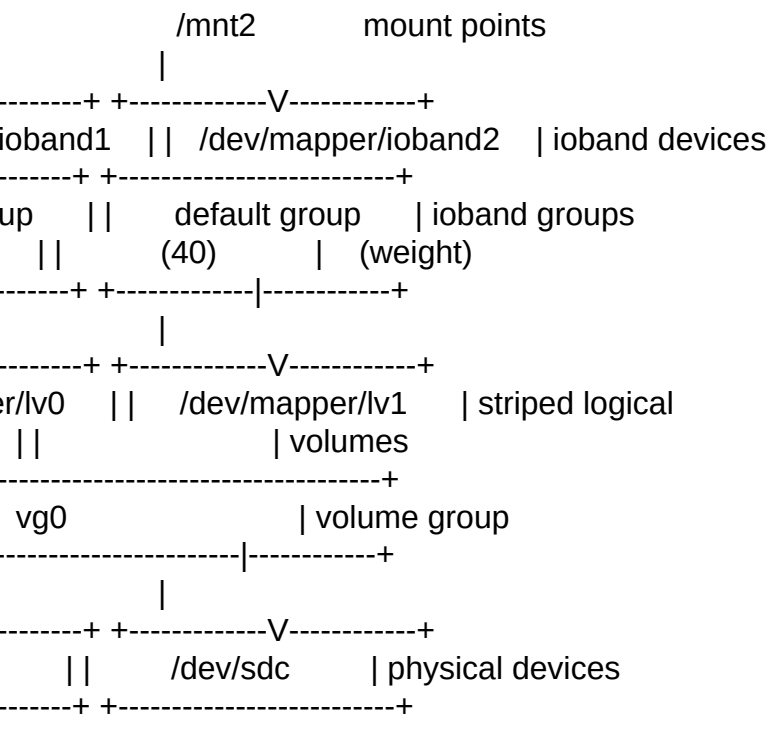
+ Display settings
+
+ SYNOPSIS
+
+ dmsetup table --target ioband
+
+ DESCRIPTION
+
+ Display the current table for the ioband device in a format. See
+ "dmsetup create" command for information on the table format.
+
+ EXAMPLE
+
+ The following output shows the current table of "ioband1."
+
+ # dmsetup table --target ioband
+ ioband: 0 32129937 ioband1 8:29 128 10 400 user weight \
+ 2048 :100 1000:80 2000:20
+
+
+ ---
+
+ Display Statistics
+
+ SYNOPSIS
+
+ dmsetup status --target ioband
+
+ DESCRIPTION
+
+ Display the statistics of all the ioband devices whose target
+ type is "ioband."
+
+ The output format is as below. the first five columns shows:
+
+ * ioband device name
+
+ * logical start sector of the device (must be 0)
+
+ * device size in sectors
+
+ * target type (must be "ioband")
+
+ * device group ID
+
+ The remaining columns show the statistics of each ioband group
+ on the band device. Each group uses seven columns for its
+ statistics.

+
+ * ioband group ID (-1 means default)
+
+ * total read requests
+
+ * delayed read requests
+
+ * total read sectors
+
+ * total write requests
+
+ * delayed write requests
+
+ * total write sectors
+
+ EXAMPLE
+
+ The following output shows the statistics of two ioband devices.
+ Ioband2 only has the default ioband group and ioband1 has three
+ (default, 1001, 1002) ioband groups.
+
+ # dmsetup status
+ ioband2: 0 44371467 ioband 128 -1 143 90 424 122 78 352
+ ioband1: 0 44371467 ioband 128 -1 223 172 408 211 136 600 1001 \
+ 166 107 472 139 95 352 1002 211 146 520 210 147 504
+
+
+ ---
+
+ Reset status counter
+
+ SYNOPSIS
+
+ dmsetup message IOBAND_DEVICE 0 reset
+
+ DESCRIPTION
+
+ Reset the statistics of ioband device IOBAND_DEVICE.
+
+ EXAMPLE
+
+ Reset the statistics of "ioband1."
+
+ # dmsetup message ioband1 0 reset
+
+
+ ---
+

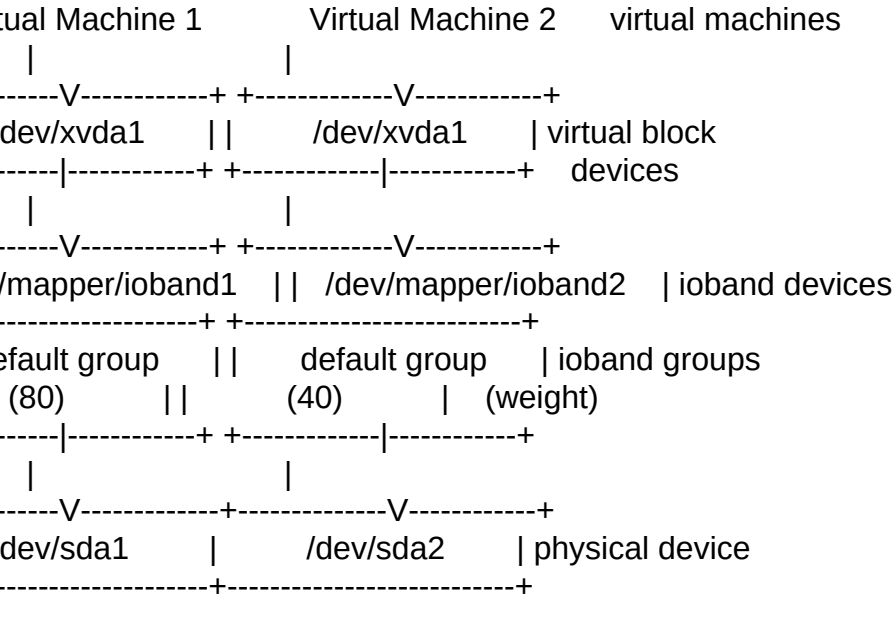
+Examples
+
+ Example #1: Bandwidth control on Partitions
+
+ This example describes how to control the bandwidth with disk
+ partitions. The following diagram illustrates the configuration of this
+ example. You may want to run a database on /dev/mapper/ioband1 and web
+ applications on /dev/mapper/ioband2.
+



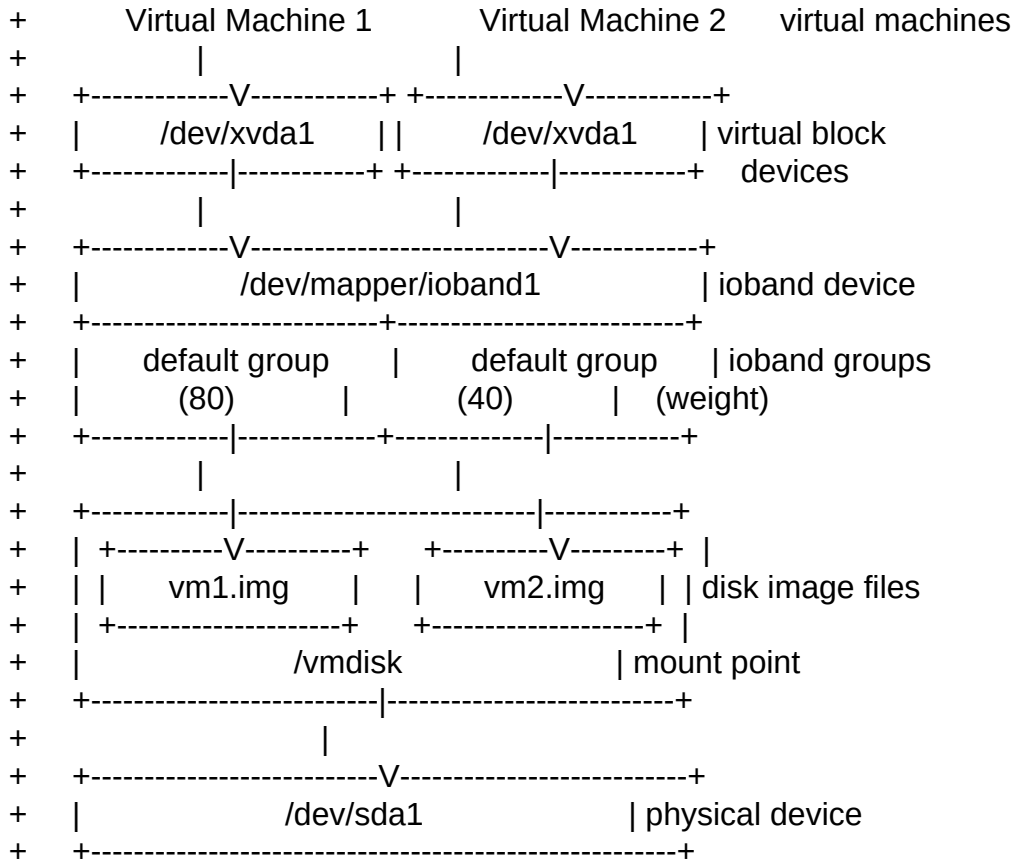
+
+
+ To setup the above configuration, follow these steps:
+
+ 1. Create ioband devices with the same device group ID and assign
+ weights of 80 and 40 to the default ioband groups respectively.
+
+ # echo "0 $(blockdev --getsize /dev/sda1) ioband /dev/sda1 1 0 0" \
+ "none weight 0 :80" | dmsetup create ioband1
+ # echo "0 $(blockdev --getsize /dev/sda2) ioband /dev/sda2 1 0 0" \
+ "none weight 0 :40" | dmsetup create ioband2
+
+
+ 2. Create filesystems on the ioband devices and mount them.
+
+ # mkfs.ext3 /dev/mapper/ioband1
+ # mount /dev/mapper/ioband1 /mnt1
+
+ # mkfs.ext3 /dev/mapper/ioband2
+ # mount /dev/mapper/ioband2 /mnt2
+
+
+ ---
+
+ Example #2: Bandwidth control on Logical Volumes
+
+ This example is similar to the example #1 but it uses LVM logical

+ volumes instead of disk partitions. This example shows how to configure
+ ioband devices on two striped logical volumes.
+
+ 
+
+
+
+ To setup the above configuration, follow these steps:
+
+ 1. Initialize the partitions for use by LVM.
+
+ # pvcreate /dev/sdb
+ # pvcreate /dev/sdc
+
+
+ 2. Create a new volume group named "vg0" with /dev/sdb and /dev/sdc.
+
+ # vgcreate vg0 /dev/sdb /dev/sdc
+
+
+ 3. Create two logical volumes in "vg0." The volumes have to be striped.
+
+ # lvcreate -n lv0 -i 2 -l 64 vg0 -L 1024M
+ # lvcreate -n lv1 -i 2 -l 64 vg0 -L 1024M
+
+
+ The rest is the same as the example #1.
+
+ 4. Create ioband devices corresponding to each logical volume and
+ assign weights of 80 and 40 to the default ioband groups respectively.
+

```
+        # echo "0 $(blockdev --getsize /dev/mapper/vg0-lv0)" \
+          "ioband /dev/mapper/vg0-lv0 1 0 0 none weight 0 :80" | \
+          dmsetup create ioband1
+        # echo "0 $(blockdev --getsize /dev/mapper/vg0-lv1)" \
+          "ioband /dev/mapper/vg0-lv1 1 0 0 none weight 0 :40" | \
+          dmsetup create ioband2
+
+
+   5.   Create filesystems on the ioband devices and mount them.
+
+        # mkfs.ext3 /dev/mapper/ioband1
+        # mount /dev/mapper/ioband1 /mnt1
+
+        # mkfs.ext3 /dev/mapper/ioband2
+        # mount /dev/mapper/ioband2 /mnt2
+
+
+  ---------------------------------------------------------------------
+
+  Example #3: Bandwidth control on processes
+
+    This example describes how to control the bandwidth with groups of
+   processes. You may also want to run an additional application on the same
+   machine described in the example #1. This example shows how to add a new
+   ioband group for this application.
+
+              /mnt1                    /mnt2           mount points
+                |                        |
+    +-------------V------------+ +-------------V------------+
+    |   /dev/mapper/ioband1    | |   /dev/mapper/ioband2    | ioband devices
+    +-------------+------------+ +-------------+------------+
+    |          default         | |  user=1000  |   default  | ioband groups
+    |           (80)           | |     (20)    |    (40)    |   (weight)
+    +-------------+------------+ +-------------+------------+
+                |                        |
+    +-------------V-------------+--------------V------------+
+    |         /dev/sda1         |          /dev/sda2        | physical device
+    +---------------------------+---------------------------+
+
+
+    The following shows to set up a new ioband group on the machine that is
+   already configured as the example #1. The application will have a weight
+   of 20 and run with user-id 1000 on /dev/mapper/ioband2.
+
+   1.   Set the type of ioband2 to "user."
+
+        # dmsetup message ioband2 0 type user.
+
```

+
+ 2. Create a new ioband group on ioband2.
+
+   # dmsetup message ioband2 0 attach 1000
+
+
+ 3. Assign weight of 10 to this newly created ioband group.
+
+   # dmsetup message ioband2 0 weight 1000:20
+
+
+ ---
+
+ Example #4: Bandwidth control for Xen virtual block devices
+
+   This example describes how to control the bandwidth for Xen virtual
+ block devices. The following diagram illustrates the configuration of this
+ example.
+
+ 
+
+
+
+   The followings shows how to map ioband device "ioband1" and "ioband2" to
+ virtual block device "/dev/xvda1 on Virtual Machine 1" and "/dev/xvda1 on
+ Virtual Machine 2" respectively on the machine configured as the example
+ #1. Add the following lines to the configuration files that are referenced
+ when creating "Virtual Machine 1" and "Virtual Machine 2."
+
+   For "Virtual Machine 1"
+   disk = ['phy:/dev/mapper/ioband1,xvda,w']
+
+   For "Virtual Machine 2"
+   disk = ['phy:/dev/mapper/ioband2,xvda,w']

+
+
+ ---
+
+ Example #5: Bandwidth control for Xen blktap devices
+
+ This example describes how to control the bandwidth for Xen virtual
+ block devices when Xen blktap devices are used. The following diagram
+ illustrates the configuration of this example.
+



+
+ To setup the above configuration, follow these steps:
+
+ 1. Create an ioband device.
+
+ # echo "0 $(blockdev --getsize /dev/sda1) ioband /dev/sda1" \
+ "1 0 0 none weight 0 :100" | dmsetup create ioband1
+
+
+ 2. Add the following lines to the configuration files that are
+ referenced when creating "Virtual Machine 1" and "Virtual Machine 2."
+ Disk image files "/vmdisk/vm1.img" and "/vmdisk/vm2.img" will be used.
+
+ For "Virtual Machine 1"

+ disk = ['tap:aio:/vmdisk/vm1.img,xvda,w',]
+
+ For "Virtual Machine 1"
+ disk = ['tap:aio:/vmdisk/vm2.img,xvda,w',]
+
+
+ 3. Run the virtual machines.
+
+ # xm create vm1
+ # xm create vm2
+
+
+ 4. Find out the process IDs of the daemons which control the blktap
+ devices.
+
+ # lsof /vmdisk/disk[12].img
+ COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
+ tapdisk 15011 root 11u REG 253,0 2147483648 48961 /vmdisk/vm1.img
+ tapdisk 15276 root 13u REG 253,0 2147483648 48962 /vmdisk/vm2.img
+
+
+ 5. Create new ioband groups of pid 15011 and pid 15276, which are
+ process IDs of the tapdisks, and assign weight of 80 and 40 to the
+ groups respectively.
+
+ # dmsetup message ioband1 0 type pid
+ # dmsetup message ioband1 0 attach 15011
+ # dmsetup message ioband1 0 weight 15011:80
+ # dmsetup message ioband1 0 attach 15276
+ # dmsetup message ioband1 0 weight 15276:40

_____

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