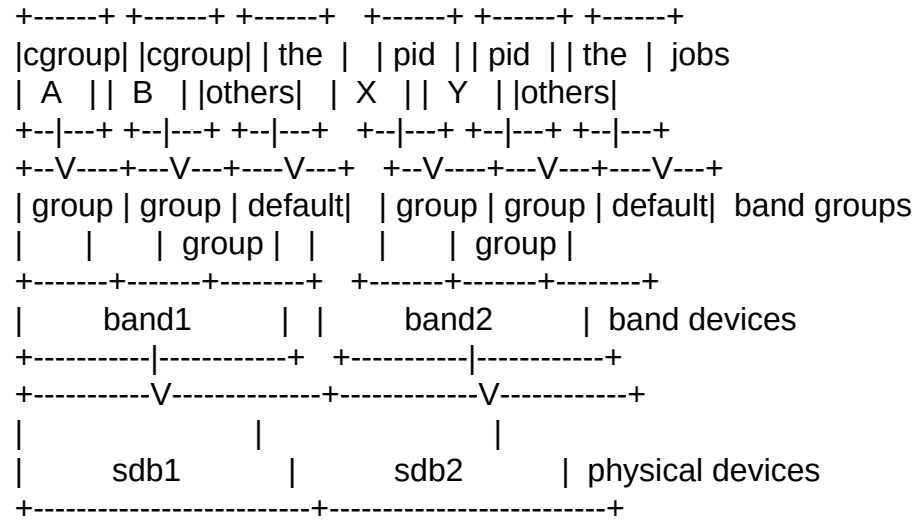
Subject: [PATCH 0/2] dm-band: The I/O bandwidth controller: Overview
Posted by Ryo Tsuruta on Wed, 23 Jan 2008 12:53:50 GMT
View Forum Message <> Reply to Message

Hi everyone,

I'm happy to announce that I've implemented a Block I/O bandwidth controller.
The controller is designed to be of use in a cgroup or virtual machine
environment. The current approach is that the controller is implemented as
a device-mapper driver.

What's dm-band all about?
==========
Dm-band 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-band 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.



How dm-band works.
==========
Every band device has one band group, which by default is called the default
group.

Band devices can also have extra band groups in them. Each band group
has a job to support and a weight. Proportional to the weight, dm-band 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. One token is consumed each
time the group passes on a request. Dm-band will refill groups with tokens
once all of groups that have requests on a given physical device use up their
tokens.

With this approach, a job running on a band group with large weight is
guaranteed to be able to issue a large number of I/O requests.

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.

You can also check the manual at Document/device-mapper/band.txt of the
linux kernel source tree for more information.

Create and map band devices

Create two band devices "band1" and "band2" and map them to "/dev/sda1"
and "/dev/sda2" respectively.

```
 # echo "0 `blockdev --getsize /dev/sda1` band /dev/sda1 1" | dmsetup create band1
 # echo "0 `blockdev --getsize /dev/sda2` band /dev/sda2 1" | dmsetup create band2
```

If the commands are successful then the device files "/dev/mapper/band1"
and "/dev/mapper/band2" will have been created.

Bandwidth control

In this example weights of 40 and 10 will be assigned to "band1" and
"band2" respectively. This is done using the following commands:

```
 # dmsetup message band1 0 weight 40
 # dmsetup message band2 0 weight 10
```

After these commands, "band1" can use 80% --- 40/(40+10)*100 --- of the
bandwidth of the physical disk "/dev/sda" while "band2" can use 20%.

Additional bandwidth control

In this example two extra band groups are created on "band1".
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.

Firstly the band group type of "band1" is set to "user".
Then, the user-id 1000 and 2000 groups are attached to "band1".
Finally, weights are assigned to the user-id 1000 and 2000 groups.

```
 # dmsetup message band1 0 type user
 # dmsetup message band1 0 attach 1000
 # dmsetup message band1 0 attach 2000
 # dmsetup message band1 0 weight 1000:30
 # dmsetup message band1 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.

Band Device	Band Group	Weight
band1	user id 1000	30
band1	user id 2000	20
band1	default group(the other users)	40
band2	default group	10

Remove band devices

Remove the band devices when no longer used.

```
  # dmsetup remove band1
  # dmsetup remove band2
```

TODO
==========
- Cgroup support.
- Control read and write requests separately.
- Support WRITE_BARRIER.
- Optimization.
- More configuration tools. Or is the dmsetup command sufficient?
- Other policies to schedule BIOs. Or is the weight policy sufficient?

Thanks,
Ryo Tsuruta

Subject: [PATCH 1/2] dm-band: The I/O bandwidth controller: Source code patch
Posted by Ryo Tsuruta on Wed, 23 Jan 2008 12:56:32 GMT
View Forum Message <> Reply to Message

Here is the patch of dm-band.

Based on 2.6.23.14
Signed-off-by: Ryo Tsuruta <ryov@valinux.co.jp>
Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```
diff -uprN linux-2.6.23.14.orig/drivers/md/Kconfig linux-2.6.23.14/drivers/md/Kconfig
--- linux-2.6.23.14.orig/drivers/md/Kconfig 2008-01-15 05:49:56.000000000 +0900
+++ linux-2.6.23.14/drivers/md/Kconfig 2008-01-21 16:09:41.000000000 +0900
@@ -276,4 +276,13 @@ config DM_DELAY

  If unsure, say N.

+config DM_BAND
+ tristate "I/O band width control "
+ depends on BLK_DEV_DM
+ ---help---
+ Any processes or cgroups can use the same storage
+ with its band-width fairly shared.
+
+ If unsure, say N.
+
 endif # MD
diff -uprN linux-2.6.23.14.orig/drivers/md/Makefile linux-2.6.23.14/drivers/md/Makefile
--- linux-2.6.23.14.orig/drivers/md/Makefile 2008-01-15 05:49:56.000000000 +0900
+++ linux-2.6.23.14/drivers/md/Makefile 2008-01-21 20:45:03.000000000 +0900
@@ -8,6 +8,7 @@ dm-multipath-objs := dm-hw-handler.o dm-
 dm-snapshot-objs := dm-snap.o dm-exception-store.o
 dm-mirror-objs := dm-log.o dm-raid1.o
 dm-rdac-objs := dm-mpath-rdac.o
+dm-band-objs := dm-bandctl.o dm-band-policy.o dm-band-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 \
@@ -39,6 +40,7 @@ obj-$(CONFIG_DM_MULTIPATH_RDAC) += dm-rd
 obj-$(CONFIG_DM_SNAPSHOT) += dm-snapshot.o
 obj-$(CONFIG_DM_MIRROR)  += dm-mirror.o
 obj-$(CONFIG_DM_ZERO)  += dm-zero.o
+obj-$(CONFIG_DM_BAND)  += dm-band.o

 quiet_cmd_unroll = UNROLL  $@
       cmd_unroll = $(PERL) $(srctree)/$(src)/unroll.pl $(UNROLL) \
diff -uprN linux-2.6.23.14.orig/drivers/md/dm-band-policy.c
linux-2.6.23.14/drivers/md/dm-band-policy.c
--- linux-2.6.23.14.orig/drivers/md/dm-band-policy.c 1970-01-01 09:00:00.000000000 +0900
```

```
+++ linux-2.6.23.14/drivers/md/dm-band-policy.c 2008-01-21 20:31:14.000000000 +0900
@@ -0,0 +1,185 @@
+/*
+ * 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-band.h"
+
+/*
+ * The following functiotons determine when and which BIOs should
+ * be submitted to control the I/O flow.
+ * It is possible to add a new I/O scheduling policy with it.
+ */
+
+
+/*
+ * Functions for weight balancing policy.
+ */
+#define DEFAULT_WEIGHT 100
+#define DEFAULT_TOKENBASE 2048
+#define BAND_IOPRIO_BASE 100
+
+static int proceed_global_epoch(struct banddevice *bs)
+{
+ bs->g_epoch++;
+#if 0 /* this will also work correct */
+ if (bs->g_blocked)
+  queue_work(bs->g_band_wq, &bs->g_conductor);
+ return 0;
+#endif
+ dprintk(KERN_ERR "proceed_epoch %d --> %d\n",
+      bs->g_epoch-1, bs->g_epoch);
+ return 1;
+}
+
+static inline int proceed_epoch(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+
+ if (bw->c_my_epoch != bs->g_epoch) {
+  bw->c_my_epoch = bs->g_epoch;
```

```
+  return 1;
+ }
+ return 0;
+}
+
+static inline int iopriority(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+ int iopri;
+
+ iopri = bw->c_token*BAND_IOPRIO_BASE/bw->c_token_init_value + 1;
+ if (bw->c_my_epoch != bs->g_epoch)
+  iopri += BAND_IOPRIO_BASE;
+ if (bw->c_going_down)
+  iopri += BAND_IOPRIO_BASE*2;
+
+ return iopri;
+}
+
+static int is_token_left(struct bandgroup *bw)
+{
+ if (bw->c_token > 0)
+  return iopriority(bw);
+
+ if (proceed_epoch(bw) || bw->c_going_down) {
+  bw->c_token = bw->c_token_init_value;
+  dprintk(KERN_ERR "refill token: bw:%p token:%d\n",
+       bw, bw->c_token);
+  return iopriority(bw);
+ }
+ return 0;
+}
+
+static void prepare_token(struct bandgroup *bw, struct bio *bio)
+{
+ bw->c_token--;
+}
+
+static void set_weight(struct bandgroup *bw, int new)
+{
+ struct banddevice *bs = bw->c_shared;
+ struct bandgroup *p;
+
+ bs->g_weight_total += (new - bw->c_weight);
+ bw->c_weight = new;
+
+ list_for_each_entry(p, &bs->g_brothers, c_list) {
+  /* Fixme: it might overflow */
```

```
+  p->c_token = p->c_token_init_value =
+     bs->g_token_base*p->c_weight/bs->g_weight_total + 1;
+ }
+}
+
+static int policy_weight_ctr(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+
+ bw->c_my_epoch = bs->g_epoch;
+ bw->c_weight = 0;
+ set_weight(bw, DEFAULT_WEIGHT);
+ return 0;
+}
+
+static void policy_weight_dtr(struct bandgroup *bw)
+{
+ set_weight(bw, 0);
+}
+
+static int policy_weight_param(struct bandgroup *bw, char *cmd, char *value)
+{
+ struct banddevice *bs = bw->c_shared;
+ int val = simple_strtol(value, NULL, 0);
+ int r = 0;
+
+ if (!strcmp(cmd, "weight")) {
+  if (val > 0)
+   set_weight(bw, val);
+  else
+   r = -EINVAL;
+ } else if (!strcmp(cmd, "token")) {
+  if (val > 0) {
+   bs->g_token_base = val;
+   set_weight(bw, bw->c_weight);
+  } else
+   r = -EINVAL;
+ } else {
+  r = -EINVAL;
+ }
+ return r;
+}
+
+/*
+ *  <Method>      <description>
+ * g_can_submit   : To determine whether a given group has a right to
+ *                  submit BIOs.
+ *       The larger return value the higher priority to submit.
```

```
+ *       Zero means it has no right.
+ * g_prepare_bio  : Called right before submitting each BIO.
+ * g_restart_bios : Called when there exist some BIOs blocked but none of them
+ *       can't be submitted now.
+ *      This method have to do something to restart to submit BIOs.
+ *       Returns 0 if it has become able to submit them now.
+ *       Otherwise, returns 1 and this policy module has to restart
+ *      sumitting BIOs by itself later on.
+ * 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 bandgroup.
+ * g_group_dtr    : Called when struct bandgroup is removed.
+ * g_set_param    : To update the policy own date.
+ *       The parameters can be passed through "dmsetup message"
+ *                 command.
+ */
+static void policy_weight_init(struct banddevice *bs)
+{
+ bs->g_can_submit = is_token_left;
+ bs->g_prepare_bio = prepare_token;
+ bs->g_restart_bios = proceed_global_epoch;
+ bs->g_group_ctr = policy_weight_ctr;
+ bs->g_group_dtr = policy_weight_dtr;
+ bs->g_set_param = policy_weight_param;
+
+ bs->g_token_base = DEFAULT_TOKENBASE;
+ bs->g_epoch = 0;
+ bs->g_weight_total = 0;
+}
+/* weight balancing policy. --- End --- */
+
+
+static void policy_default_init(struct banddevice *bs) /*XXX*/
+{
+ policy_weight_init(bs); /* temp */
+}
+
+struct policy_type band_policy_type[] = {
+ {"default", policy_default_init},
+ {"weight", policy_weight_init},
+ {NULL,     policy_default_init}
+};
diff -uprN linux-2.6.23.14.orig/drivers/md/dm-band-type.c
linux-2.6.23.14/drivers/md/dm-band-type.c
--- linux-2.6.23.14.orig/drivers/md/dm-band-type.c 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.23.14/drivers/md/dm-band-type.c 2008-01-21 20:27:15.000000000 +0900
@@ -0,0 +1,69 @@
```

```
+/*
+ * 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-band.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 band_process_id(struct bio *bio)
+{
+ /*
+  * This function will work for KVM and Xen.
+  */
+ return (int)current->tgid;
+}
+
+static int band_process_group(struct bio *bio)
+{
+ return (int)process_group(current);
+}
+
+static int band_uid(struct bio *bio)
+{
+ return (int)current->uid;
+}
+
+static int band_cpuset(struct bio *bio)
+{
+ return 0; /* not implemented yet */
+}
+
+static int band_node(struct bio *bio)
+{
+ return 0; /* not implemented yet */
+}
+
```

```
+static int band_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 band_group_type[] = {
+ {"none",   NULL},
+ {"pgrp",   band_process_group},
+ {"pid",    band_process_id},
+ {"node",   band_node},
+ {"cpuset", band_cpuset},
+ {"cgroup", band_cgroup},
+ {"user",   band_uid},
+ {NULL,     NULL}
+};
diff -uprN linux-2.6.23.14.orig/drivers/md/dm-band.h linux-2.6.23.14/drivers/md/dm-band.h
--- linux-2.6.23.14.orig/drivers/md/dm-band.h 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.23.14/drivers/md/dm-band.h 2008-01-21 20:20:54.000000000 +0900
@@ -0,0 +1,99 @@
+/*
+ * Copyright (C) 2008 VA Linux Systems Japan K.K.
+ *
+ *  I/O bandwidth control
+ *
+ * This file is released under the GPL.
+ */
+
+#define DEFAULT_IO_THROTTLE 4
+#define DEFAULT_IO_LIMIT 128
+#define BAND_NAME_MAX 31
+#define BAND_ID_ANY (-1)
+
+struct bandgroup;
+
+struct banddevice {
+ struct list_head g_brothers;
+ struct work_struct g_conductor;
+ struct workqueue_struct *g_band_wq;
+ int g_io_throttle;
+ int g_io_limit;
+ int g_plug_bio;
+ int g_issued;
```

```
+ int g_blocked;
+ spinlock_t g_lock;
+
+ int g_devgroup;
+ int g_ref;  /* just for debugging */
+ struct list_head g_list;
+ int g_flags; /* rfu */
+ char g_name[BAND_NAME_MAX + 1]; /* rfu */
+
+ /* policy dependent */
+ int (*g_can_submit)(struct bandgroup *);
+ void (*g_prepare_bio)(struct bandgroup *, struct bio *);
+ int (*g_restart_bios)(struct banddevice *);
+ void (*g_hold_bio)(struct bandgroup *, struct bio *);
+ struct bio * (*g_pop_bio)(struct bandgroup *);
+ int (*g_group_ctr)(struct bandgroup *);
+ void (*g_group_dtr)(struct bandgroup *);
+ int (*g_set_param)(struct bandgroup *, char *cmd, char *value);
+
+ /* members for weight balancing policy */
+ int g_epoch;
+ int g_weight_total;
+ int g_token_base;
+
+};
+
+struct bandgroup_stat {
+ unsigned long sectors;
+ unsigned long immediate;
+ unsigned long deferred;
+};
+
+struct bandgroup {
+ struct list_head c_list;
+ struct banddevice *c_shared;
+ struct dm_dev *c_dev;
+ int c_going_down;
+ struct bio_list c_blocked_bios;
+ struct list_head c_children;
+ int c_id; /* should be unsigned long or unsigned long long */
+ char c_name[BAND_NAME_MAX + 1]; /* rfu */
+ int c_issued;
+ int c_blocked;
+ struct bandgroup_stat c_stat[2]; /* hold rd/wr status */
+
+ /* type dependent */
+ int (*c_getid)(struct bio *);
+
```

```
+ /* members for weight balancing policy */
+ int c_weight;
+ int c_my_epoch;
+ int c_token;
+ int c_token_init_value;
+
+ /* rfu */
+ /* struct bio_list c_ordered_tag_bios; */
+};
+
+struct policy_type {
+ const char *p_name;
+ void   (*p_policy_init)(struct banddevice *);
+};
+
+extern struct policy_type band_policy_type[];
+
+struct group_type {
+ const char *t_name;
+ int   (*t_getid)(struct bio *);
+};
+
+extern struct group_type band_group_type[];
+
+/* Just for debugging */
+extern int band_debug;
+#define dprintk(format, a...) \
+ if (band_debug > 0) band_debug--, printk(format, ##a)
diff -uprN linux-2.6.23.14.orig/drivers/md/dm-bandctl.c linux-2.6.23.14/drivers/md/dm-bandctl.c
--- linux-2.6.23.14.orig/drivers/md/dm-bandctl.c 1970-01-01 09:00:00.000000000 +0900
+++ linux-2.6.23.14/drivers/md/dm-bandctl.c 2008-01-23 20:48:19.000000000 +0900
@@ -0,0 +1,833 @@
+/*
+ * 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 "dm.h"
+#include "dm-bio-list.h"
```

```
+#include "dm-band.h"
+
+#define DM_MSG_PREFIX "band"
+
+static LIST_HEAD(banddevice_list);
+static DEFINE_SPINLOCK(banddevicelist_lock);  /* to protect banddevice_list */
+
+static void band_conduct(struct work_struct *);
+static void band_hold_bio(struct bandgroup *, struct bio *);
+static struct bio *band_pop_bio(struct bandgroup *);
+
+int band_debug = 0; /* just for debugging */
+
+static void policy_init(struct banddevice *bs, char *name)
+{
+ struct policy_type *p;
+ for (p = band_policy_type; (p->p_name); p++) {
+  if (!strcmp(name, p->p_name))
+   break;
+ }
+
+ p->p_policy_init(bs);
+ if (!bs->g_hold_bio)
+  bs->g_hold_bio = band_hold_bio;
+ if (!bs->g_pop_bio)
+  bs->g_pop_bio = band_pop_bio;
+}
+
+static struct banddevice *alloc_banddevice(int devgroup_id, char *name,
+    int io_throttle, int io_limit)
+{
+ struct banddevice *bs = NULL;
+ struct banddevice *p;
+ struct banddevice *new;
+ unsigned long flags;
+
+ new = kzalloc(sizeof(struct banddevice), 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_band_wq = create_workqueue("kband");
+ if (!new->g_band_wq) {
+  kfree(new);
+  new = NULL;
```

```
+  goto try_to_find;
+ }
+
+ INIT_WORK(&new->g_conductor, band_conduct);
+ INIT_LIST_HEAD(&new->g_brothers);
+ INIT_LIST_HEAD(&new->g_list);
+ spin_lock_init(&new->g_lock);
+ new->g_devgroup = devgroup_id;
+ new->g_io_throttle = io_throttle;
+ new->g_io_limit = io_limit;
+ new->g_plug_bio = 0;
+ new->g_issued = 0;
+ new->g_blocked = 0;
+ new->g_ref = 0;
+ new->g_flags = 0;
+ memset(new->g_name, 0, sizeof(new->g_name));
+ new->g_hold_bio = NULL;
+ new->g_pop_bio = NULL;
+
+try_to_find:
+ spin_lock_irqsave(&banddevicelist_lock, flags);
+ list_for_each_entry(p, &banddevice_list, g_list) {
+  if (p->g_devgroup == devgroup_id) {
+   bs = p;
+   break;
+  }
+ }
+ if (!bs && (new)) {
+  policy_init(new, name);
+  bs = new;
+  new = NULL;
+  list_add_tail(&bs->g_list, &banddevice_list);
+ }
+ spin_unlock_irqrestore(&banddevicelist_lock, flags);
+
+ if (new) {
+  destroy_workqueue(new->g_band_wq);
+  kfree(new);
+ }
+
+ return bs;
+}
+
+static inline void release_banddevice(struct banddevice *bs)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&banddevicelist_lock, flags);
```

```
+ if (!list_empty(&bs->g_brothers)) {
+  spin_unlock_irqrestore(&banddevicelist_lock, flags);
+  return;
+ }
+ list_del(&bs->g_list);
+ spin_unlock_irqrestore(&banddevicelist_lock, flags);
+ destroy_workqueue(bs->g_band_wq);
+ kfree(bs);
+}
+
+static struct bandgroup *bandgroup_find(struct bandgroup *parent, int id)
+{
+ struct bandgroup *p;
+ struct bandgroup *bw = NULL;
+
+ list_for_each_entry(p, &parent->c_children, c_children) {
+  if (p->c_id == id || id == BAND_ID_ANY)
+   bw = p;
+ }
+ return bw;
+}
+
+static int bandgroup_init(struct bandgroup *bw,
+  struct bandgroup *parent, struct banddevice *bs, int id)
+{
+ unsigned long flags;
+
+ INIT_LIST_HEAD(&bw->c_list);
+ bio_list_init(&bw->c_blocked_bios);
+ bw->c_id = id; /* should be verified */
+ bw->c_going_down = 0;
+ bw->c_issued = 0;
+ bw->c_blocked = 0;
+ memset(bw->c_stat, 0, sizeof(bw->c_stat));
+
+ INIT_LIST_HEAD(&bw->c_children);
+
+ bw->c_shared = bs;
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+ if (bandgroup_find(bw, id)) {
+  spin_unlock_irqrestore(&bs->g_lock, flags);
+  DMWARN("bandgroup: id=%d already exists.\n", id);
+  return -EEXIST;
+ }
+ bs->g_ref++;
+ list_add_tail(&bw->c_list, &bs->g_brothers);
+
```

```
+ bs->g_group_ctr(bw);
+
+ if (parent) {
+  list_add_tail(&bw->c_children, &parent->c_children);
+  bw->c_dev = parent->c_dev;
+ }
+
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+
+ return 0;
+}
+
+static inline void bandgroup_release(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+
+ list_del(&bw->c_list);
+ list_del(&bw->c_children);
+ bs->g_ref--;
+ bs->g_group_dtr(bw);
+ kfree(bw);
+}
+
+static void bandgroup_destroy_all(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+ struct bandgroup *child;
+ unsigned long flags;
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+ while ((child = bandgroup_find(bw, BAND_ID_ANY)))
+  bandgroup_release(child);
+ bandgroup_release(bw);
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+}
+
+static void bandgroup_stop(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+ unsigned long flags;
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+ bw->c_going_down = 1;
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+ queue_work(bs->g_band_wq, &bs->g_conductor);
+ flush_scheduled_work();
+}
+
```

```
+static void bandgroup_stop_all(struct bandgroup *parent)
+{
+ struct banddevice *bs = parent->c_shared;
+ struct bandgroup *p;
+ unsigned long flags;
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+ list_for_each_entry(p, &parent->c_children, c_children)
+  p->c_going_down = 1;
+ parent->c_going_down = 1;
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+ queue_work(bs->g_band_wq, &bs->g_conductor);
+ flush_scheduled_work();
+ /* should wait for bw->c_issued becoming zero? */
+}
+
+static void bandgroup_resume_all(struct bandgroup *parent)
+{
+ struct banddevice *bs = parent->c_shared;
+ struct bandgroup *p;
+ unsigned long flags;
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+ list_for_each_entry(p, &parent->c_children, c_children)
+  p->c_going_down = 0;
+ parent->c_going_down = 0;
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+}
+
+/*
+ * Create a new band device:
+ *   parameters:  <device> <device-group-id> [<io_throttle>] [<io_limit>]
+ */
+static int band_ctr(struct dm_target *ti, unsigned int argc, char **argv)
+{
+ struct bandgroup *bw;
+ struct banddevice *bs;
+ int io_throttle = DEFAULT_IO_THROTTLE;
+ int io_limit = DEFAULT_IO_LIMIT;
+ int devgroup_id;
+ int val;
+ int r = 0;
+
+ if (argc < 2) {
+  ti->error = "Requires 2 or more arguments";
+  return -EINVAL;
+ }
+
```

```
+ bw = kzalloc(sizeof(struct bandgroup), GFP_KERNEL);
+ if (!bw) {
+  ti->error = "Cannot allocate memory for bandgroup";
+  return -ENOMEM;
+ }
+
+ val = simple_strtol(argv[1], NULL, 0);
+ if (val < 0) {
+  ti->error = "Device Group ID # is too large";
+  r = -EINVAL;
+  goto error;
+ }
+ devgroup_id = val;
+ dprintk(KERN_ERR "band_ctr device group id:%d\n", val);
+
+ if (argc >= 3) {
+  val = simple_strtol(argv[2], NULL, 0);
+  if (val > 0) io_throttle = val;
+  dprintk(KERN_ERR "band_ctr ioqueue_low:%d\n", io_throttle);
+ }
+ if (argc >= 4) {
+  val = simple_strtol(argv[3], NULL, 0);
+  if (val > 0) io_limit = val;
+  dprintk(KERN_ERR "band_ctr ioqueue_high:%d\n", io_limit);
+ }
+ if (io_limit < io_throttle)
+  io_limit = io_throttle;
+
+ if (dm_get_device(ti, argv[0], 0, ti->len,
+     dm_table_get_mode(ti->table), &bw->c_dev)) {
+  ti->error = "band: device lookup failed";
+  r = -EINVAL;
+  goto error;
+ }
+
+ bs = alloc_banddevice(devgroup_id, "default", io_throttle, io_limit);
+ if (!bs) {
+  ti->error = "Cannot allocate memory for banddevice";
+  r = -ENOMEM;
+  goto error2;
+ }
+
+ bandgroup_init(bw, NULL, bs, BAND_ID_ANY);
+ bw->c_getid = band_group_type[0].t_getid;
+ strlcpy(bw->c_name, band_group_type[0].t_name, sizeof(bw->c_name));
+
+ ti->private = bw;
+
```

```
+ return 0;
+
+error2:
+ dm_put_device(ti, bw->c_dev);
+error:
+ kfree(bw);
+ return r;
+}
+
+static void band_dtr(struct dm_target *ti)
+{
+ struct bandgroup *bw = ti->private;
+ struct banddevice *bs = bw->c_shared;
+
+ bandgroup_stop_all(bw);
+ dm_put_device(ti, bw->c_dev);
+ bandgroup_destroy_all(bw);
+ release_banddevice(bs);
+}
+
+static void band_hold_bio(struct bandgroup *bw, struct bio *bio)
+{
+ /* Todo: The list should be split into a read list and a write list */
+ bio_list_add(&bw->c_blocked_bios, bio);
+}
+
+static struct bio *band_pop_bio(struct bandgroup *bw)
+{
+ return bio_list_pop(&bw->c_blocked_bios);
+}
+
+static inline void prepare_to_issue(struct bandgroup *bw, struct bio *bio)
+{
+ struct banddevice *bs = bw->c_shared;
+
+ bs->g_prepare_bio(bw, bio);
+ bs->g_issued++;
+ bw->c_issued++;
+ if (bs->g_issued >= bs->g_io_limit)
+  bs->g_plug_bio = 1;
+}
+
+static inline int room_for_bio(struct bandgroup *bw)
+{
+ struct banddevice *bs = bw->c_shared;
+
+ return !bs->g_plug_bio || bw->c_going_down;
+}
```

```
+
+static inline void hold_bio(struct bandgroup *bw, struct bio *bio)
+{
+ struct banddevice *bs = bw->c_shared;
+
+ bs->g_blocked++;
+ bw->c_blocked++;
+ bw->c_stat[bio_data_dir(bio)].deferred++;
+ bs->g_hold_bio(bw, bio);
+}
+
+static inline int release_bios(struct bandgroup *bw, struct bio_list *bl)
+{
+ struct banddevice *bs = bw->c_shared;
+ struct bio *bio;
+
+ while (bs->g_can_submit(bw) && bw->c_blocked) {
+  if (!room_for_bio(bw))
+   return 1;
+  bio = bs->g_pop_bio(bw);
+  if (!bio)
+   return 0;
+#if 0
+  if (bs->g_blocked == bs->g_io_limit*2)
+   wake_up(&bs->waitq_io);
+#endif
+  bs->g_blocked--;
+  bw->c_blocked--;
+  prepare_to_issue(bw, bio);
+  bio_list_add(bl, bio);
+ }
+
+ return 0;
+}
+
+static inline struct bandgroup *bandgroup_get(
+   struct bandgroup *parent, struct bio *bio)
+{
+ struct bandgroup *bw;
+
+ if (!parent->c_getid)
+  return parent;
+
+ bw = bandgroup_find(parent, parent->c_getid(bio));
+
+ if (!bw)
+  bw = parent;
+ return bw;
```

```
+}
+
+/*
+ * Start to control the bandwidth once the number of uncompleted BIOs
+ * exceeds the value of "io_throttle".
+ */
+static int band_map(struct dm_target *ti, struct bio *bio,
+       union map_info *map_context)
+{
+ struct bandgroup *bw = ti->private;
+ struct bandgroup_stat *bws;
+ struct banddevice *bs = bw->c_shared;
+ unsigned long flags;
+
+#if 0 /* not supported yet */
+ if (bio_barrier(bio))
+  return -EOPNOTSUPP;
+#endif
+
+ /*
+  * ToDo: Block to accept new requests when the number of the blocked
+  *       BIOs becomes extremely large.
+  */
+ bio->bi_bdev = bw->c_dev->bdev;
+ bio->bi_sector -= ti->begin;
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+#if 0
+ /* ToDo: Should only stop processes requesting too many BIOs. */
+ wait_event_lock_irq(&bs->waitq_io, bs->g_blocked < bs->g_io_limit*2,
+      bs->g_lock, do_nothing);
+#endif
+ bw = bandgroup_get(bw, bio);
+ bws = &bw->c_stat[bio_data_dir(bio)];
+ bws->sectors += bio_sectors(bio);
+retry:
+ if (!bw->c_blocked && room_for_bio(bw)) {
+  if (bs->g_can_submit(bw)) {
+   bws->immediate++;
+   prepare_to_issue(bw, bio);
+   spin_unlock_irqrestore(&bs->g_lock, flags);
+   return 1;
+  } else if (bs->g_issued < bs->g_io_throttle) {
+   dprintk(KERN_ERR "band_map: token expired "
+     "bw:%p bio:%p\n", bw, bio);
+   if (bs->g_restart_bios(bs))
+    goto retry;
+  }
```

```
+ }
+ hold_bio(bw, bio);
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+
+ return 0;
+}
+
+/*
+ * Select the best group to resubmit its BIOs.
+ */
+static inline struct bandgroup *choose_best_group(struct banddevice *bs)
+{
+ struct bandgroup *bw;
+ struct bandgroup *best = NULL;
+ int highest = 0;
+ int pri;
+
+ /* Todo: The algorithm should be optimized.
+  *       It would be better to use rbtree.
+  */
+ list_for_each_entry(bw, &bs->g_brothers, c_list) {
+  if (!bw->c_blocked || !room_for_bio(bw))
+   continue;
+  pri = bs->g_can_submit(bw);
+  if (pri > highest) {
+   highest = pri;
+   best = bw;
+  }
+ }
+
+ 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 band_conduct(struct work_struct *work)
+{
+ struct banddevice *bs =
+  container_of(work, struct banddevice, g_conductor);
+ struct bandgroup *bw = NULL;
+ struct bio *bio;
+ unsigned long flags;
+ struct bio_list bl;
+
+ bio_list_init(&bl);
+
```

```
+ spin_lock_irqsave(&bs->g_lock, flags);
+retry:
+ while (bs->g_blocked && (bw = choose_best_group(bs)))
+  release_bios(bw, &bl);
+
+ if (bs->g_blocked && bs->g_issued < bs->g_io_throttle) {
+  dprintk(KERN_ERR "band_conduct: token expired bw:%p\n", bw);
+  if (bs->g_restart_bios(bs))
+   goto retry;
+ }
+
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+
+ while ((bio = bio_list_pop(&bl)))
+  generic_make_request(bio);
+}
+
+static int band_end_io(struct dm_target *ti, struct bio *bio,
+       int error, union map_info *map_context)
+{
+ struct bandgroup *bw = ti->private;
+ struct banddevice *bs = bw->c_shared;
+ unsigned long flags;
+
+ /* Todo: The algorithm should be optimized to eliminate the spinlock. */
+ spin_lock_irqsave(&bs->g_lock, flags);
+ if (bs->g_issued <= bs->g_io_throttle)
+  bs->g_plug_bio = 0;
+ bs->g_issued--;
+ bw->c_issued--;
+
+ /*
+  * Todo: It would be better to introduce high/low water marks here
+  *   not to kick the workqueues so often.
+  */
+ if (bs->g_blocked && !bs->g_plug_bio)
+  queue_work(bs->g_band_wq, &bs->g_conductor);
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+ return 0;
+}
+
+static void band_presuspend(struct dm_target *ti)
+{
+ struct bandgroup *bw = ti->private;
+ bandgroup_stop_all(bw);
+}
+
+static void band_resume(struct dm_target *ti)
```

```
+{
+ struct bandgroup *bw = ti->private;
+ bandgroup_resume_all(bw);
+}
+
+
+static void bandgroup_status(struct bandgroup *bw, int *szp, char *result,
+    unsigned int maxlen)
+{
+ struct bandgroup_stat *bws;
+ unsigned long reqs;
+ int i, sz = *szp; /* used in DMEMIT() */
+
+ if (bw->c_id == BAND_ID_ANY)
+  DMEMIT("\n default");
+ else
+  DMEMIT("\n %d", bw->c_id);
+ for (i = 0; i < 2; i++) {
+  bws = &bw->c_stat[i];
+  reqs = bws->immediate + bws->deferred;
+  DMEMIT(" %lu %lu %lu",
+   bws->immediate + bws->deferred, bws->deferred,
+   bws->sectors);
+ }
+ *szp = sz;
+}
+
+static int band_status(struct dm_target *ti, status_type_t type,
+   char *result, unsigned int maxlen)
+{
+ struct bandgroup *bw = ti->private, *p;
+ struct banddevice *bs = bw->c_shared;
+ int sz = 0; /* used in DMEMIT() */
+ unsigned long flags;
+
+ switch (type) {
+ case STATUSTYPE_INFO:
+  spin_lock_irqsave(&bs->g_lock, flags);
+  DMEMIT("devgrp=%u # read-req delay sect write-req delay sect",
+         bs->g_devgroup);
+  bandgroup_status(bw, &sz, result, maxlen);
+  list_for_each_entry(p, &bw->c_children, c_children)
+   bandgroup_status(p, &sz, result, maxlen);
+  spin_unlock_irqrestore(&bs->g_lock, flags);
+  break;
+
+ case STATUSTYPE_TABLE:
+  DMEMIT("%s devgrp=%u io_throttle=%u io_limit=%u",
```

```
+     bw->c_dev->name, bs->g_devgroup,
+     bs->g_io_throttle, bs->g_io_limit);
+  DMEMIT("\n id=default type=%s weight=%u token=%u",
+   bw->c_name, bw->c_weight, bw->c_token_init_value);
+  list_for_each_entry(p, &bw->c_children, c_children)
+   DMEMIT("\n id=%d weight=%u token=%u",
+    p->c_id, p->c_weight, p->c_token_init_value);
+  break;
+ }
+
+ return 0;
+}
+
+static int band_group_type_select(struct bandgroup *bw, char *name)
+{
+ struct banddevice *bs = bw->c_shared;
+ struct group_type *t;
+ unsigned long flags;
+
+ for (t = band_group_type; (t->t_name); t++) {
+  if (!strcmp(name, t->t_name))
+   break;
+ }
+ if (!t->t_name) {
+  DMWARN("band tyep select: %s isn't supported.\n", name);
+  return -EINVAL;
+ }
+ spin_lock_irqsave(&bs->g_lock, flags);
+ if (!list_empty(&bw->c_children)) {
+  spin_unlock_irqrestore(&bs->g_lock, flags);
+  return -EBUSY;
+ }
+ bw->c_getid = t->t_getid;
+ strlcpy(bw->c_name, t->t_name, sizeof(bw->c_name));
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+
+ return 0;
+}
+
+static inline int split_string(char *s, char **v)
+{
+ int id = 0;
+ char *p, *q;
+
+ p = strsep(&s, "=:,");
+ q = strsep(&s, "=:,");
+ if (!q) {
+  *v = p;
```

```
+ } else {
+  id = simple_strtol(p, NULL, 0);
+  *v = q;
+ }
+ return id;
+}
+
+static int band_set_param(struct bandgroup *bw, char *cmd, char *value)
+{
+ struct banddevice *bs = bw->c_shared;
+ char *val_str;
+ int id;
+ unsigned long flags;
+ int r;
+
+ id = split_string(value, &val_str);
+
+ spin_lock_irqsave(&bs->g_lock, flags);
+ if (id) {
+  bw = bandgroup_find(bw, id);
+  if (!bw) {
+   spin_unlock_irqrestore(&bs->g_lock, flags);
+   DMWARN("band_set_param: id=%d not found.\n", id);
+   return -EINVAL;
+  }
+ }
+ r = bs->g_set_param(bw, cmd, val_str);
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+ return r;
+}
+
+static int band_group_attach(struct bandgroup *bw, int id)
+{
+ struct banddevice *bs = bw->c_shared;
+ struct bandgroup *sub_bw;
+ int r;
+
+ if (id <= 0) {
+  DMWARN("band_group_attach: invalid id:%d\n", id);
+  return -EINVAL;
+ }
+ sub_bw = kzalloc(sizeof(struct bandgroup), GFP_KERNEL);
+ if (!sub_bw)
+  return -ENOMEM;
+
+ r = bandgroup_init(sub_bw, bw, bs, id);
+ if (r < 0) {
+  kfree(sub_bw);
```

```
+  return r;
+ }
+ return 0;
+}
+
+static int band_group_detach(struct bandgroup *bw, int id)
+{
+ struct banddevice *bs = bw->c_shared;
+ struct bandgroup *sub_bw;
+ unsigned long flags;
+
+ if (id <= 0) {
+  DMWARN("band_group_detach: invalid id:%d\n", id);
+  return -EINVAL;
+ }
+ spin_lock_irqsave(&bs->g_lock, flags);
+ sub_bw = bandgroup_find(bw, id);
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+ if (!sub_bw) {
+  DMWARN("band_group_detach: invalid id:%d\n", id);
+  return -EINVAL;
+ }
+ bandgroup_stop(sub_bw);
+ spin_lock_irqsave(&bs->g_lock, flags);
+ bandgroup_release(sub_bw);
+ spin_unlock_irqrestore(&bs->g_lock, flags);
+ return 0;
+}
+
+/*
+ * Message parameters:
+ * "policy"      <name>
+ *       ex)
+ *  "policy" "weight"
+ * "type"        "none"|"pid"|"pgrp"|"node"|"cpuset"|"cgroup"|"user"
+ *  "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 band_message(struct dm_target *ti, unsigned int argc, char **argv)
+{
+ struct bandgroup *bw = ti->private, *p;
+ struct banddevice *bs = bw->c_shared;
```

```
+ int val = 0;
+ int r = 0;
+ unsigned long flags;
+
+ if (argc == 1 && !strcmp(argv[0], "reset")) {
+  spin_lock_irqsave(&bs->g_lock, flags);
+  memset(bw->c_stat, 0, sizeof(bw->c_stat));
+  list_for_each_entry(p, &bw->c_children, c_children)
+   memset(p->c_stat, 0, sizeof(p->c_stat));
+  spin_unlock_irqrestore(&bs->g_lock, flags);
+  return 0;
+ }
+
+ if (argc != 2) {
+  DMWARN("Unrecognised band message received.");
+  return -EINVAL;
+ }
+ if (!strcmp(argv[0], "debug")) {
+  band_debug = simple_strtol(argv[1], NULL, 0);
+  if (band_debug < 0) band_debug = 0;
+  return 0;
+ } else if (!strcmp(argv[0], "io_throttle")) {
+  val = simple_strtol(argv[1], NULL, 0);
+  spin_lock_irqsave(&bs->g_lock, flags);
+  if (val > 0 && val <= bs->g_io_limit)
+   bs->g_io_throttle = val;
+  spin_unlock_irqrestore(&bs->g_lock, flags);
+  return 0;
+ } else if (!strcmp(argv[0], "io_limit")) {
+  val = simple_strtol(argv[1], NULL, 0);
+  spin_lock_irqsave(&bs->g_lock, flags);
+  if (val > bs->g_io_throttle)
+   bs->g_io_limit = val;
+  spin_unlock_irqrestore(&bs->g_lock, flags);
+  return 0;
+ } else if (!strcmp(argv[0], "type")) {
+  return band_group_type_select(bw, argv[1]);
+ } else if (!strcmp(argv[0], "attach")) {
+  /* message attach <group-id> */
+  int id = simple_strtol(argv[1], NULL, 0);
+  return band_group_attach(bw, id);
+ } else if (!strcmp(argv[0], "detach")) {
+  /* message detach <group-id> */
+  int id = simple_strtol(argv[1], NULL, 0);
+  return band_group_detach(bw, id);
+ } else {
+  /* message anycommand <group-id>:<value> */
+  r = band_set_param(bw, argv[0], argv[1]);
```

```
+  if (r < 0)
+   DMWARN("Unrecognised band message received.");
+  return r;
+ }
+ return 0;
+}
+
+static struct target_type band_target = {
+ .name      = "band",
+ .module      = THIS_MODULE,
+ .version     = {0, 0, 2},
+ .ctr      = band_ctr,
+ .dtr      = band_dtr,
+ .map      = band_map,
+ .end_io      = band_end_io,
+ .presuspend  = band_presuspend,
+ .resume      = band_resume,
+ .status      = band_status,
+ .message     = band_message,
+};
+
+static int __init dm_band_init(void)
+{
+ int r;
+
+ r = dm_register_target(&band_target);
+ if (r < 0) {
+  DMERR("register failed %d", r);
+  return r;
+ }
+ return r;
+}
+
+static void __exit dm_band_exit(void)
+{
+ int r;
+
+ r = dm_unregister_target(&band_target);
+ if (r < 0)
+  DMERR("unregister failed %d", r);
+}
+
+module_init(dm_band_init);
+module_exit(dm_band_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");

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

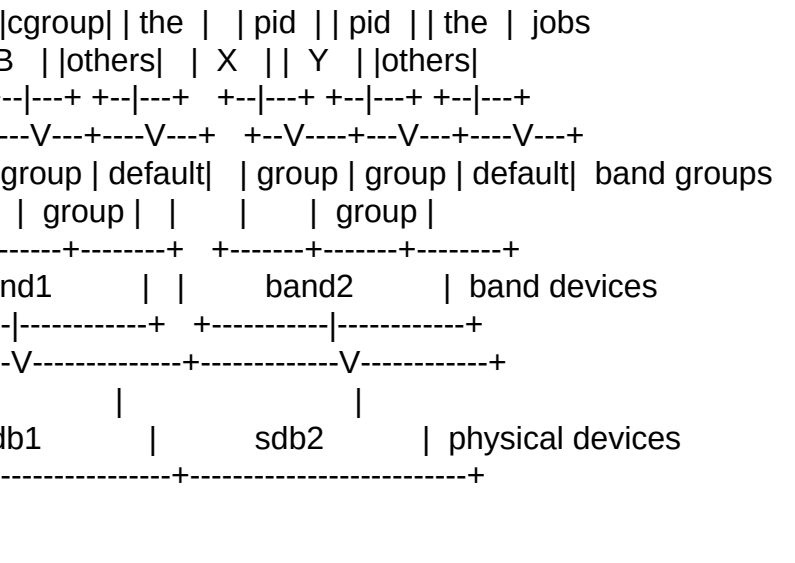
---

Subject: [PATCH 2/2] dm-band: The I/O bandwidth controller: Document
Posted by Ryo Tsuruta on Wed, 23 Jan 2008 12:58:44 GMT
View Forum Message <> Reply to Message

Here is the document of dm-band.

Based on 2.6.23.14
Signed-off-by: Ryo Tsuruta <ryov@valinux.co.jp>
Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

diff -uprN linux-2.6.23.14.orig/Documentation/device-mapper/band.txt
linux-2.6.23.14/Documentation/device-mapper/band.txt
--- linux-2.6.23.14.orig/Documentation/device-mapper/band.txt 1970-01-01 09:00:00.000000000
+0900
+++ linux-2.6.23.14/Documentation/device-mapper/band.txt 2008-01-23 21:48:46.000000000
+0900
@@ -0,0 +1,431 @@
+====================
+Document for dm-band
+====================
+
+Contents:
+ What's dm-band all about?
+ How dm-band works
+ Setup and Installation
+ Command Reference
+ TODO
+
+
+What's dm-band all about?
+=========================
+Dm-band 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-band 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.
+
+ +------+ +------+ +------+ +------+ +------+ +------+



How dm-band works.
==========

Every band device has one band group, which by default is called the default group.

Band devices can also have extra band groups in them. Each band group has a job to support and a weight. Proportional to the weight, dm-band 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. One token is consumed each time the group passes on a request. Dm-band will refill groups with tokens once all of groups that have requests on a given physical device use up their tokens.

With this approach, a job running on a band group with large weight is guaranteed to be able to issue a large number of I/O requests.

Setup and Installation
==========

Build a kernel with these options enabled:

```
CONFIG_MD
CONFIG_BLK_DEV_DM
CONFIG_DM_BAND
```

If compiled as module, use modprobe to load dm-band.

```
# make modules
# make modules_install
```

```
+  # depmod -a
+  # modprobe dm-band
+
+"dmsetup targets" command shows all available device-mapper targets.
+"band" is displayed if dm-band has loaded.
+
+  # dmsetup targets
+  band            v0.0.2
+
+
+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 band devices
+---------------------------
+Create two band devices "band1" and "band2" and map them to "/dev/sda1"
+and "/dev/sda2" respectively.
+
+ # echo "0 `blockdev --getsize /dev/sda1` band /dev/sda1 1" | dmsetup create band1
+ # echo "0 `blockdev --getsize /dev/sda2` band /dev/sda2 1" | dmsetup create band2
+
+If the commands are successful then the device files "/dev/mapper/band1"
+and "/dev/mapper/band2" will have been created.
+
+
+Bandwidth control
+-----------------
+In this example weights of 40 and 10 will be assigned to "band1" and
+"band2" respectively. This is done using the following commands:
+
+ # dmsetup message band1 0 weight 40
+ # dmsetup message band2 0 weight 10
+
+After these commands, "band1" can use 80% --- 40/(40+10)*100 --- of the
+bandwidth of the physical disk "/dev/sda" while "band2" can use 20%.
+
+
+Additional bandwidth control
+----------------------------
+In this example two extra band groups are created on "band1".
+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.
+
```

+Firstly the band group type of "band1" is set to "user".
+Then, the user-id 1000 and 2000 groups are attached to "band1".
+Finally, weights are assigned to the user-id 1000 and 2000 groups.
+
+ # dmsetup message band1 0 type user
+ # dmsetup message band1 0 attach 1000
+ # dmsetup message band1 0 attach 2000
+ # dmsetup message band1 0 weight 1000:30
+ # dmsetup message band1 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.
+
+ Band Device   Band Group   Weight
+ band1   user id 1000   30
+ band1   user id 2000   20
+ band1   default group(the other users)   40
+ band2   default group   10
+
+
+Remove band devices
+------------------
+Remove the band devices when no longer used.
+
+ # dmsetup remove band1
+ # dmsetup remove band2
+
+
+Command Reference
+================
+
+
+Create a band device
+-------------------
+SYNOPSIS
+ dmsetup create BAND_DEVICE
+
+DESCRIPTION
+ The following space delimited arguments, which describe the physical device
+ may are read from standard input. All arguments are required, and they must
+ be provided in order the order listed below.
+
+   starting sector of the physical device
+   size in sectors of the physical device
+   string "band" as a target type
+   physical device name
+   device group ID
+

```
+  You must set the same device group ID for each band device that shares
+  the same bandwidth.
+
+  A default band group is also created and attached to the band device.
+
+  If the command is successful, the device file
+  "/dev/device-mapper/BAND_DEVICE" will have been created.
+
+EXAMPLE
+  Create a band device with the following parameters:
+    physical device = "/dev/sda1"
+    band device name = "band1"
+    device group ID = "100"
+
+    # size=`blockdev --getsize /dev/sda1`
+    # echo "0 $size band /dev/sda1 100" | dmsetup create band1
+
+  Create two device groups (ID=1,2). The bandwidth of each device group may be
+  individually controlled.
+
+    # echo "0 11096883 band /dev/sda1 1" | dmsetup create band1
+    # echo "0 11096883 band /dev/sda2 1" | dmsetup create band2
+    # echo "0 11096883 band /dev/sda3 2" | dmsetup create band3
+    # echo "0 11096883 band /dev/sda4 2" | dmsetup create band4
+
+
+Remove the band device
+----------------------
+SYNOPSIS
+  dmsetup remove BAND_DEVICE
+
+DESCRIPTION
+  Remove the band device with the given name. All band groups that are attached
+  to the band device are removed automatically.
+
+EXAMPLE
+  Remove the band device "band1".
+
+  # dmsetup remove band1
+
+
+Set a band group's type
+-----------------------
+SYNOPSIS
+  dmsetup message BAND_DEVICE 0 type TYPE
+
+DESCRIPTION
+  Set a band group's type. TYPE must be one of "user", "pid" or "pgrp".
```

+
+EXAMPLE
+ Set a band group's type to "user".
+
+ # dmsetup message band1 0 type user
+
+
+Create a band group
+------------------
+SYNOPSIS
+ dmsetup message BAND_DEVICE 0 attach ID
+
+DESCRIPTION
+ Create a band group and attach it a band device. The ID number specifies the
+ user-id, pid or pgrp, as per the the type.
+
+EXAMPLE
+ Attach a band group with uid 1000 to the band device "band1".
+
+ # dmsetup message band1 0 type user
+ # dmsetup message band1 0 attach 1000
+
+
+Remove a band group
+------------------
+SYNOPSIS
+ dmsetup message BAND_DEVICE 0 detach ID
+
+DESCRIPTION
+ Detach a band group specified by ID from a band device.
+
+EXAMPLE
+ Detach the band group with ID "2000" from the band device "band2".
+
+ # dmsetup message band2 0 detach 1000
+
+
+Set the weight of a band group
+------------------------------
+SYNOPSIS
+ dmsetup message BAND_DEVICE 0 weight VAL
+ dmsetup message BAND_DEVICE 0 weight ID:VAL
+
+DESCRIPTION
+ Set the weight of band group. The weight is evaluated as a ratio against the
+ total weight. The following example means that "band1" can use 80% ---
+ 40/(40+10)*100 --- of the bandwidth of the physical disk "/dev/sda" while
+ "band2" can use 20%.

```
+
+    # dmsetup message band1 0 weight 40
+    # dmsetup message band1 0 weight 10
+
+  The following has the same effect as the above commands:
+
+    # dmsetup message band1 0 weight 4
+    # dmsetup message band2 0 weight 1
+
+  VAL must be an integer grater than 0. The default is 100.
+
+EXAMPLE
+  Set the weight of the default band group to 40.
+
+  # dmsetup message band1 0 weight 40
+
+  Set the weight of the band group with ID "1000" to 10.
+
+  # dmsetup message band1 0 weight 1000:10
+
+
+Set the number of tokens
+------------------------
+SYNOPSIS
+  dmsetup message BAND_DEVICE 0 token VAL
+
+DESCRIPTION
+  Set the number of tokens. The value is applied to the all band devices
+  that have the same device group ID as BAND_DEVICE.
+  VAL must be an integer grater than 0. The default is 2048.
+
+EXAMPLE
+  Set a token to 256.
+
+  # dmsetup message band1 0 token 256
+
+
+Set I/O throttling
+------------------
+SYNOPSIS
+  dmsetup message BAND_DEVICE 0 io_throttle VAL
+
+DESCRIPTION
+  Set I/O throttling. The value is applied to all band devices that have the
+  same device group ID as BAND_DEVICE.
+  VAL must be an integer grater than 0. The default is 4.
+
+  I/O requests are throttled up until the number of in-progress I/Os reaches
```

+ this value.
+
+EXAMPLE
+ Set I/O throttling to 16.
+
+ # dmsetup message band1 0 io_throttle 16
+
+
+Set I/O limiting
+----------------
+SYNOPSIS
+ dmsetup message BAND_DEVICE 0 io_limit VAL
+
+DESCRIPTION
+ Set I/O limiting. The value is applied to the all band devices that have
+ the same device group ID as BAND_DEVICE.
+ VAL must be an integer grater than 0. The default is 128.
+
+ When the number of in-progress I/Os reaches this value, subsequent I/O
+ requests are blocked.
+
+EXAMPLE
+ Set an io_limit to 128.
+
+ # dmsetup message band1 0 io_limit 128
+
+
+Display settings
+----------------
+SYNOPSIS
+ dmsetup table --target band
+
+DESCRIPTION
+ Display the settings of each band device.
+
+ The output format is as below:
+   On the first line for a device, space delimited.
+   Band device name
+   Starting sector of partition
+   Partition size in sectors
+   Target type
+   Device number (major:minor)
+   Device group ID
+   I/O throttle
+   I/O limit
+
+   On subsequent indented lines for a device, space delimited.
+   Group ID

```
+      Group type
+      Weight
+      Token
+
+EXAMPLE
+  # dmsetup table --target band
+  band2: 0 11096883 band 8:30 devgrp=0 io_throttle=4 io_limit=128
+   id=default type=none weight=20 token=205
+  band1: 0 11096883 band 8:31 devgrp=0 io_throttle=4 io_limit=128
+   id=default type=user weight=80 token=820
+   id=1000 weight=80 token=820
+   id=2000 weight=20 token=205
+
+
+Display Statistics
+------------------
+SYNOPSIS
+  dmsetup status --target band
+
+DESCRIPTION
+  Display the statistics of each band device.
+
+  The output format is as below:
+    On the first line for a device, space delimited.
+      Band Device Name
+      Start Sector of Device
+      Device Size in Sectors
+      Target Type
+      Device Group ID
+
+    On subsequent indented lines for a device, space delimited.
+      "parent" or Group ID,
+      Total read requests
+      Delayed read requests
+      Total read sectors
+      Total write requests
+      Delayed write requests
+      Total write sectors
+
+EXAMPLE
+  # dmsetup status
+    band2: 0 11096883 band devgrp=0 # read-req delay sect write-req delay sect
+     parent 913 898 7304 899 886 7192
+    band1: 0 11096883 band devgrp=0 # read-req delay sect write-req delay sect
+     parent 121 100 968 101 85 808
+     1000 482 468 3856 491 473 3928
+     2000 502 489 4016 469 448 3752
+
```

```
+
+Reset status counter
+--------------------
+SYNOPSIS
+  dmsetup message BAND_DEVICE 0 reset
+
+DESCRIPTION
+  Reset the status counter of a band device.
+
+EXAMPLE
+  Reset the "band1" counter.
+
+  # dmsetup message band1 0 reset
+
+
+TODO
+=======================
+  - Cgroup support.
+  - Control read and write requests separately.
+  - Support WRITE_BARRIER.
+  - Optimization.
+  - More configuration tools. Or is the dmsetup command sufficient?
+  - Other policies to schedule BIOs. Or is the weight policy sufficient?
```

_____

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

---

Subject: Re: [PATCH 0/2] dm-band: The I/O bandwidth controller: Overview
Posted by Anthony Liguori on Wed, 23 Jan 2008 19:22:36 GMT
View Forum Message <> Reply to Message

Hi,

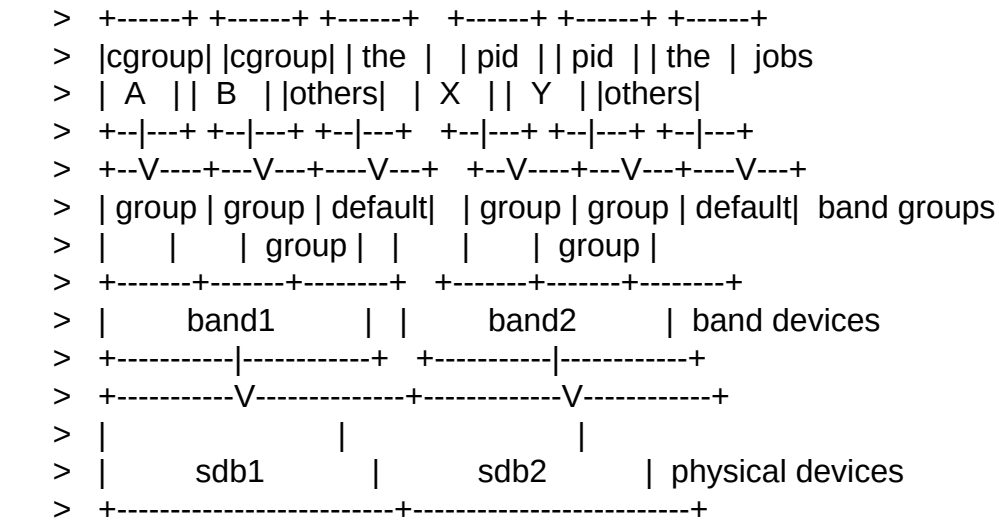
I believe this work is very important especially in the context of virtual machines. I think it would be more useful though implemented in the context of the IO scheduler. Since we already support a notion of IO priority, it seems reasonable to add a notion of an IO cap.

Regards,

Anthony Liguori

Ryo Tsuruta wrote:
> Hi everyone,
>
> I'm happy to announce that I've implemented a Block I/O bandwidth controller.

> The controller is designed to be of use in a cgroup or virtual machine
> environment. The current approach is that the controller is implemented as
> a device-mapper driver.
>
> What's dm-band all about?
> ========================
> Dm-band 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-band 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.
>



>
>
> How dm-band works.
> ========================
> Every band device has one band group, which by default is called the default
> group.
>
> Band devices can also have extra band groups in them. Each band group
> has a job to support and a weight. Proportional to the weight, dm-band 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. One token is consumed each
> time the group passes on a request. Dm-band will refill groups with tokens
> once all of groups that have requests on a given physical device use up their
> tokens.
>
> With this approach, a job running on a band group with large weight is

> guaranteed to be able to issue a large number of I/O requests.
>
>
> 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.
>
> You can also check the manual at Document/device-mapper/band.txt of the
> linux kernel source tree for more information.
>
>
> Create and map band devices
> ---------------------------
> Create two band devices "band1" and "band2" and map them to "/dev/sda1"
> and "/dev/sda2" respectively.
>
> # echo "0 `blockdev --getsize /dev/sda1` band /dev/sda1 1" | dmsetup create band1
> # echo "0 `blockdev --getsize /dev/sda2` band /dev/sda2 1" | dmsetup create band2
>
> If the commands are successful then the device files "/dev/mapper/band1"
> and "/dev/mapper/band2" will have been created.
>
>
> Bandwidth control
> ----------------
> In this example weights of 40 and 10 will be assigned to "band1" and
> "band2" respectively. This is done using the following commands:
>
> # dmsetup message band1 0 weight 40
> # dmsetup message band2 0 weight 10
>
> After these commands, "band1" can use 80% --- 40/(40+10)*100 --- of the
> bandwidth of the physical disk "/dev/sda" while "band2" can use 20%.
>
>
> Additional bandwidth control
> ---------------------------
> In this example two extra band groups are created on "band1".
> 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.
>
> Firstly the band group type of "band1" is set to "user".
> Then, the user-id 1000 and 2000 groups are attached to "band1".
> Finally, weights are assigned to the user-id 1000 and 2000 groups.
>

> # dmsetup message band1 0 type user
> # dmsetup message band1 0 attach 1000
> # dmsetup message band1 0 attach 2000
> # dmsetup message band1 0 weight 1000:30
> # dmsetup message band1 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.
>
> Band Device   Band Group   Weight
>   band1   user id 1000   30
>   band1   user id 2000   20
>   band1   default group(the other users)   40
>   band2   default group   10
>
>
> Remove band devices
> ------------------
> Remove the band devices when no longer used.
>
>   # dmsetup remove band1
>   # dmsetup remove band2
>
>
> TODO
> ========================
>   - Cgroup support.
>   - Control read and write requests separately.
>   - Support WRITE_BARRIER.
>   - Optimization.
>   - More configuration tools. Or is the dmsetup command sufficient?
>   - Other policies to schedule BIOs. Or is the weight policy sufficient?
>
> Thanks,
> Ryo Tsuruta

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

---

Subject: Re: [PATCH 2/2] dm-band: The I/O bandwidth controller: Document
Posted by Andi Kleen on Wed, 23 Jan 2008 19:57:57 GMT
View Forum Message <> Reply to Message

Ryo Tsuruta <ryov@valinux.co.jp> writes:

> Here is the document of dm-band.

Could you please address in the document how the intended use
cases/feature set etc. differs from CFQ2 io priorities?

Thanks,

-Andi

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

---

Subject: Re: [PATCH 0/2] dm-band: The I/O bandwidth controller: Overview
Posted by Hirokazu Takahashi on Thu, 24 Jan 2008 08:11:45 GMT
View Forum Message <> Reply to Message

Hi,

> Hi,
>
> I believe this work is very important especially in the context of
> virtual machines. I think it would be more useful though implemented in
> the context of the IO scheduler. Since we already support a notion of
> IO priority, it seems reasonable to add a notion of an IO cap.

I agree that what you proposed is the most straightforward approach.
Ryo and I have also investigated the CFQ scheduler that It will be possible
to enhance it to support bandwidth control with quite a few modification.
I think both approach have pros and cons.

At this time, we have chosen the device-mapper approach because:
 - it can work with any I/O scheduler. Some people will want use the NOOP
   scheduler against high-end storages.
 - only people who need I/O bandwidth control should use it.
 - it is independent to the I/O schedulers so that it will be easy to maintain.
 - it can keep the CFQ implementation simple.

The current the CFQ scheduler has some limitations if you want to control the
bandwidths. The scheduler only has seven priority levels, which also means it
has only seven classes. If you assign the same io-priority A to several VMs
--- virtual machines ---, these machines have to share the I/O bandwidth which
is assign to the io-priority A class. If other VM with io-priority B which is
lower than io-priority A and there is no other VM in the same io-priority B
class, VM in the io-priority B class may be able to use large bandwidth than
VMs in io-priority the A class.

I guess two level scheduling should be introduced in the CFQ scheduler if needed.
The one is to choose the best cgroup or job, and the other is to choose the
highest io-priority class.

There is another limitation that io-priority is global so that it affects
all the disks to access. It isn't allowed to have a job use several io-priorities
to access several disks respectively. I think "per disk io-priority" will be
required.

But the device-mapper approach also has a bad points.
It is hard to get the capabilities and configurations of the underlying devices
such as information of partitions or LUNs. So some configuration tools may
probably be required.

Thank you,
Hirokazu Takahashi.

> Regards,
>
> Anthony Liguori
>
> Ryo Tsuruta wrote:
> > Hi everyone,
> >
> > I'm happy to announce that I've implemented a Block I/O bandwidth controller.
> > The controller is designed to be of use in a cgroup or virtual machine
> > environment. The current approach is that the controller is implemented as
> > a device-mapper driver.
> >
> > What's dm-band all about?
> > ========================
> > Dm-band 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-band 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.
> >
> > +------+ +------+ +------+   +------+ +------+ +------+
> > |cgroup| |cgroup| | the | | pid | | pid | | the | jobs
> > | A | | B | |others| | X | | Y | |others|
> > +--|---+ +--|---+ +--|---+   +--|---+ +--|---+ +--|---+
> > +--V----+---V---+----V---+   +--V----+---V---+----V---+
> > | group | group | default| | group | group | default| band groups
> > | | | group | | | | group |
> > +-------+-------+--------+   +-------+-------+--------+

> > | band1 | | band2 | band devices
> > +-----------|------------+ +-----------|------------+
> > +-----------V--------------+-------------V------------+
> > | | |
> > | sdb1 | sdb2 | physical devices
> > +-------------------------+------------------------+
> >
> >
> > How dm-band works.
> > ======================
> > Every band device has one band group, which by default is called the default
> > group.
> >
> > Band devices can also have extra band groups in them. Each band group
> > has a job to support and a weight. Proportional to the weight, dm-band 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. One token is consumed each
> > time the group passes on a request. Dm-band will refill groups with tokens
> > once all of groups that have requests on a given physical device use up their
> > tokens.
> >
> > With this approach, a job running on a band group with large weight is
> > guaranteed to be able to issue a large number of I/O requests.
> >
> >
> > 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.
> >
> > You can also check the manual at Document/device-mapper/band.txt of the
> > linux kernel source tree for more information.
> >
> >
> > Create and map band devices
> > ---------------------------
> > Create two band devices "band1" and "band2" and map them to "/dev/sda1"
> > and "/dev/sda2" respectively.
> >
> > # echo "0 `blockdev --getsize /dev/sda1` band /dev/sda1 1" | dmsetup create band1
> > # echo "0 `blockdev --getsize /dev/sda2` band /dev/sda2 1" | dmsetup create band2
> >
> > If the commands are successful then the device files "/dev/mapper/band1"
> > and "/dev/mapper/band2" will have been created.

> >
> >
> > Bandwidth control
> > ----------
> > In this example weights of 40 and 10 will be assigned to "band1" and
> > "band2" respectively. This is done using the following commands:
> >
> > # dmsetup message band1 0 weight 40
> > # dmsetup message band2 0 weight 10
> >
> > After these commands, "band1" can use 80% --- 40/(40+10)*100 --- of the
> > bandwidth of the physical disk "/dev/sda" while "band2" can use 20%.
> >
> >
> > Additional bandwidth control
> > ---------------------------
> > In this example two extra band groups are created on "band1".
> > 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.
> >
> > Firstly the band group type of "band1" is set to "user".
> > Then, the user-id 1000 and 2000 groups are attached to "band1".
> > Finally, weights are assigned to the user-id 1000 and 2000 groups.
> >
> > # dmsetup message band1 0 type user
> > # dmsetup message band1 0 attach 1000
> > # dmsetup message band1 0 attach 2000
> > # dmsetup message band1 0 weight 1000:30
> > # dmsetup message band1 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.
> >
> > Band Device   Band Group   Weight
> >   band1   user id 1000   30
> >   band1   user id 2000   20
> >   band1   default group(the other users)   40
> >   band2   default group   10
> >
> >
> > Remove band devices
> > -------------------
> > Remove the band devices when no longer used.
> >
> >   # dmsetup remove band1
> >   # dmsetup remove band2
> >

> >
> > TODO
> > ========================
> > - Cgroup support.
> > - Control read and write requests separately.
> > - Support WRITE_BARRIER.
> > - Optimization.
> > - More configuration tools. Or is the dmsetup command sufficient?
> > - Other policies to schedule BIOs. Or is the weight policy sufficient?
> >
> > Thanks,
> > Ryo Tsuruta
>
> --
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

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

---

Subject: Re: [PATCH 2/2] dm-band: The I/O bandwidth controller: Document
Posted by Ryo Tsuruta on Thu, 24 Jan 2008 10:32:02 GMT
View Forum Message <> Reply to Message

Hi,

> > Here is the document of dm-band.
>
> Could you please address in the document how the intended use
> cases/feature set etc. differs from CFQ2 io priorities?

Thank you for your suggestion, I'll do that step by step.

Thanks,
Ryo Tsuruta

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

---

Subject: dm-band: The I/O bandwidth controller: Performance Report
Posted by Ryo Tsuruta on Fri, 25 Jan 2008 07:07:20 GMT

Hi,

Now I report the result of dm-band bandwidth control test I did yesterday.
I've got really good results that dm-band works as I expected. I made
several band-groups on several disk partitions and gave them heavy I/O loads.

Hardware Spec.
==============
  DELL Dimention E521:

  Linux kappa.local.valinux.co.jp 2.6.23.14 #1 SMP
    Thu Jan 24 17:24:59 JST 2008 i686 athlon i386 GNU/Linux
  Detected 2004.217 MHz processor.
  CPU0: AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ stepping 02
  Memory: 966240k/981888k available (2102k kernel code, 14932k reserved,
    890k data, 216k init, 64384k highmem)
  scsi 2:0:0:0: Direct-Access   ATA   ST3250620AS   3.AA PQ: 0 ANSI: 5
  sd 2:0:0:0: [sdb] 488397168 512-byte hardware sectors (250059 MB)
  sd 2:0:0:0: [sdb] Write Protect is off
  sd 2:0:0:0: [sdb] Mode Sense: 00 3a 00 00
  sd 2:0:0:0: [sdb] Write cache: enabled, read cache: enabled,
    doesn't support DPO or FUA
  sdb: sdb1 sdb2 < sdb5 sdb6 sdb7 sdb8 sdb9 sdb10 sdb11 sdb12 sdb13 sdb14
    sdb15 >

The results of bandwidth control test on partitions
===

The configurations of the test #1:
   o Prepare three partitions sdb5, sdb6 and sdb7.
   o Give weights of 40, 20 and 10 to sdb5, sdb6 and sdb7 respectively.
   o Run 128 processes issuing random read/write direct I/O with 4KB data
     on each device at the same time.
   o Count up the number of I/Os and sectors which have done in 60 seconds.

                The result of the test #1

| device | sdb5 | sdb6 | sdb7 |
| weight | 40 (57.0%) | 20 (29.0%) | 10 (14.0%) |
|-----------------+-------------------+-------------------+-----------------|
I/Os (r/w)	6640(3272/ 3368)	3434(1719/ 1715)	1689(857/ 832)
sectors (r/w)	53120(26176/26944)	27472(13752/13720)	13512(6856/6656)
ratio to total	56.4%	29.2%	14.4%

The configurations of the test #2:

o The configurations are the same as the test #1 except this test doesn't
run any processes issuing I/Os on sdb6.

The result of the test #2

device	sdb5	sdb6	sdb7
weight	40 (57.0%)	20 (29.0%)	10 (14.0%)
I/Os (r/w)	9566(4815/ 4751)	0(0/ 0)	2370(1198/1172)
sectors (r/w)	76528(38520/38008)	0(0/ 0)	18960(9584/9376)
ratio to total	76.8%	0.0%	23.2%

The results of bandwidth control test on band-groups.
==========
The configurations of the test #3:
o Prepare three partitions sdb5 and sdb6.
o Create two extra band-groups on sdb5, the first is of user1 and the
second is of user2.
o Give weights of 40, 20, 10 and 10 to the user1 band-group, the user2
band-group, the default group of sdb5 and sdb6 respectively.
o Run 128 processes issuing random read/write direct I/O with 4KB data
on each device at the same time.
o Count up the number of I/Os and sectors which have done in 60 seconds.

The result of the test #3

dev	sdb5			sdb6
usr	user1	user2	other users	all users
wgt	40 (50.0%)	20 (25.0%)	10 (12.5%)	10 (12.5%)
I/O	5951(2940/ 3011)	3068(1574/ 1494)	1663(828/ 835)	1663(810/ 853)
sec	47608(23520/24088)	24544(12592/11952)	13304(6624/6680)	13304(6480/6824)
%	48.2%	24.9%	13.5%	13.5%

The configurations of the test #4:
o The configurations are the same as the test #3 except this test doesn't
run any processes issuing I/Os on the user2 band-group.

The result of the test #4

dev	sdb5			sdb6
usr	user1	user2	other users	all users
wgt	40 (50.0%)	20 (25.0%)	10 (12.5%)	10 (12.5%)

|I/O| 8002(3963/ 4039)| 0(0/   0)| 2056(1021/1035)| 2008(998/1010)|
|sec|64016(31704/32312)| 0(0/   0)|16448(8168/8280)|16064(7984/8080)|
| % | 66.3% | 0.0% | 17.0% | 16.6% |

Conclusions and future works
==========================
Dm-band works well with random I/Os. I have a plan on running some tests
using various real applications such as databases or file servers.
If you have any other good idea to test dm-band, please let me know.

Thank you,
Ryo Tsuruta.

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

---

Subject: Re: [Xen-devel] dm-band: The I/O bandwidth controller: Performance Report
Posted by INAKOSHI Hiroya on Tue, 29 Jan 2008 06:42:17 GMT
View Forum Message <> Reply to Message

Hi,

Ryo Tsuruta wrote:
> The results of bandwidth control test on band-groups.
> ==
> The configurations of the test #3:
>   o Prepare three partitions sdb5 and sdb6.
>   o Create two extra band-groups on sdb5, the first is of user1 and the
>   second is of user2.
>   o Give weights of 40, 20, 10 and 10 to the user1 band-group, the user2
>   band-group, the default group of sdb5 and sdb6 respectively.
>   o Run 128 processes issuing random read/write direct I/O with 4KB data
>   on each device at the same time.

you mean that you run 128 processes on each user-device pairs? Namely,
I guess that

  user1: 128 processes on sdb5,
  user2: 128 processes on sdb5,
  another: 128 processes on sdb5,
  user2: 128 processes on sdb6.

> Conclusions and future works
> ==========================

> Dm-band works well with random I/Os. I have a plan on running some tests
> using various real applications such as databases or file servers.
> If you have any other good idea to test dm-band, please let me know.

The second preliminary studies might be:

- What if you use a different I/O size on each device (or device-user pair)?
- What if you use a different number of processes on each device (or device-user pair)?

And my impression is that it's natural dm-band is in device-mapper, separated from I/O scheduler. Because bandwidth control and I/O scheduling are two different things, it may be simpler that they are implemented in different layers.

Regards,

Hiroya.

>
> Thank you,
> Ryo Tsuruta.
>
> _____
> Xen-devel mailing list
> Xen-devel@lists.xensource.com
> http://lists.xensource.com/xen-devel
>
>

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