dm-switch

The device-mapper switch target creates a device that supports an arbitrary mapping of fixed-size regions of I/O across a fixed set of paths. The path used for any specific region can be switched dynamically by sending the target a message.

It maps I/O to underlying block devices efficiently when there is a large number of fixed-sized address regions but there is no simple pattern that would allow for a compact representation of the mapping such as dm-stripe.

Background

Dell EqualLogic and some other iSCSI storage arrays use a distributed frameless architecture. In this architecture, the storage group consists of a number of distinct storage arrays ("members") each having independent controllers, disk storage and network adapters. When a LUN is created it is spread across multiple members. The details of the spreading are hidden from initiators connected to this storage system. The storage group exposes a single target discovery portal, no matter how many members are being used. When iSCSI sessions are created, each session is connected to an eth port on a single member. Data to a LUN can be sent on any iSCSI session, and if the blocks being accessed are stored on another member the I/O will be forwarded as required. This forwarding is invisible to the initiator. The storage layout is also dynamic, and the blocks stored on disk may be moved from member to member as needed to balance the load.

This architecture simplifies the management and configuration of both the storage group and initiators. In a multipathing configuration, it is possible to set up multiple iSCSI sessions to use multiple network interfaces on both the host and target to take advantage of the increased network bandwidth. An initiator could use a simple round robin algorithm to send I/O across all paths and let the storage array members forward it as necessary, but there is a performance advantage to sending data directly to the correct member.

A device-mapper table already lets you map different regions of a device onto different targets. However in this architecture the LUN is spread with an address region size on the order of 10s of MBs, which means the resulting table could have more than a million entries and consume far too much memory.

Using this device-mapper switch target we can now build a two-layer device hierarchy:

> Upper Tier - Determine which array member the I/O should be sent to. Lower Tier - Load balance amongst paths to a particular member.

The lower tier consists of a single dm multipath device for each member. Each of these multipath devices contains the set of paths directly to the array member in one priority group, and leverages existing path selectors to load balance amongst these paths. We also build a non-preferred priority group containing paths to other array members for failover reasons.

The upper tier consists of a single dm-switch device. This device uses a bitmap to look up the location of the I/O and choose the appropriate lower tier device to route the I/O. By using a bitmap we are able to use 4 bits for each address range in a 16 member group (which is very large for us). This is a much denser representation than the dm table b-tree can achieve.

Construction Parameters

<num_paths> <region_size> <num_optional_args> [<optional_args>...] [<dev_path> <offset>]+

<num_paths>
: The number of paths across which to distribute the I/O.

<region_size>
: The number of 512-byte sectors in a region. Each region can be redirected to any of the available paths.

<num_optional_args>
: The number of optional arguments. Currently, no optional arguments are supported and so this must be zero.

<dev_path>
: The block device that represents a specific path to the device.

<offset>
: The offset of the start of data on the specific <dev_path> (in units of 512-byte sectors). This number is added to the sector number when forwarding the request to the specific path. Typically it is zero.

Messages

set_region_mappings <index>:<path_nr> [<index>]:<path_nr> [<index>]:<path_nr>...

Modify the region table by specifying which regions are redirected to which paths.

<index>
: The region number (region size was specified in constructor parameters). If index is omitted, the next region (previous index + 1) is used. Expressed in hexadecimal (WITHOUT any prefix like 0x).

<path_nr>
: The path number in the range 0 ... (<num_paths> - 1). Expressed in hexadecimal (WITHOUT any prefix like 0x).

R<n>,<m>

This parameter allows repetitive patterns to be loaded quickly. <n> and <m> are hexadecimal numbers. The last <n> mappings are repeated in the next <m> slots.

Status

No status line is reported.

Example

Assume that you have volumes vg1/switch0 vg1/switch1 vg1/switch2 with the same size.

Create a switch device with 64kB region size:

```
dmsetup create switch --table "0 `blockdev --getsz /dev/vg1/switch0`
    switch 3 128 0 /dev/vg1/switch0 0 /dev/vg1/switch1 0 /dev/vg1/switch2 0"
```

Set mappings for the first 7 entries to point to devices switch0, switch1, switch2, switch0, switch1, switch2, switch1:

```
dmsetup message switch 0 set_region_mappings 0:0 :1 :2 :0 :1 :2 :1
```

Set repetitive mapping. This command:

```
dmsetup message switch 0 set_region_mappings 1000:1 :2 R2,10
```

is equivalent to:

```
dmsetup message switch 0 set_region_mappings 1000:1 :2 :1 :2 :1 :2 :1 :2 \
    :1 :2 :1 :2 :1 :2 :1 :2 :1 :2
```