Device-mapper "unstriped" target

Introduction

The device-mapper "unstriped" target provides a transparent mechanism to unstripe a device-mapper "striped" target to access the underlying disks without having to touch the true backing block-device. It can also be used to unstripe a hardware RAID-0 to access backing disks.

Parameters: <number of stripes> <chunk size> <stripe #> <dev_path> <offset>

<number of stripes>
: The number of stripes in the RAID 0.

<chunk size>
: The amount of 512B sectors in the chunk striping.

<dev_path>
: The block device you wish to unstripe.

<stripe #>
: The stripe number within the device that corresponds to physical drive you wish to unstripe. This must be 0 indexed.

Why use this module?

An example of undoing an existing dm-stripe

This small bash script will setup 4 loop devices and use the existing striped target to combine the 4 devices into one. It then will use the unstriped target ontop of the striped device to access the individual backing loop devices. We write data to the newly exposed unstriped devices and verify the data written matches the correct underlying device on the striped array:

```
#!/bin/bash

MEMBER_SIZE=$((128 * 1024 * 1024))
NUM=4
SEQ_END=$((${NUM}-1))
CHUNK=256
BS=4096

RAID_SIZE=$((${MEMBER_SIZE}*${NUM}/512))
DM_PARMS="0 ${RAID_SIZE} striped ${NUM} ${CHUNK}"
COUNT=$((${MEMBER_SIZE} / ${BS}))

for i in $(seq 0 ${SEQ_END}); do
  dd if=/dev/zero of=member-${i} bs=${MEMBER_SIZE} count=1 oflag=direct
  losetup /dev/loop${i} member-${i}
  DM_PARMS+=" /dev/loop${i} 0"
done

echo $DM_PARMS | dmsetup create raid0
for i in $(seq 0 ${SEQ_END}); do
  echo "0 1 unstriped ${NUM} ${CHUNK} ${i} /dev/mapper/raid0 0" | dmsetup create set-${i}
done;

for i in $(seq 0 ${SEQ_END}); do
  dd if=/dev/urandom of=/dev/mapper/set-${i} bs=${BS} count=${COUNT} oflag=direct
  diff /dev/mapper/set-${i} member-${i}
done;

for i in $(seq 0 ${SEQ_END}); do
  dmsetup remove set-${i}
done

dmsetup remove raid0

for i in $(seq 0 ${SEQ_END}); do
  losetup -d /dev/loop${i}
  rm -f member-${i}
done
```

Another example

Intel NVMe drives contain two cores on the physical device. Each core of the drive has segregated access to its LBA range. The current LBA model has a RAID 0 128k chunk on each core, resulting in a 256k stripe across the two cores:

```
 Core 0:       Core 1:
__________    __________
```

```
| LBA 512|     | LBA 768|
| LBA 0  |     | LBA 256|
----------     ----------
```

The purpose of this unstriping is to provide better QoS in noisy neighbor environments. When two partitions are created on the aggregate drive without this unstriping, reads on one partition can affect writes on another partition. This is because the partitions are striped across the two cores. When we unstripe this hardware RAID 0 and make partitions on each new exposed device the two partitions are now physically separated.

With the dm-unstriped target we're able to segregate an fio script that has read and write jobs that are independent of each other. Compared to when we run the test on a combined drive with partitions, we were able to get a 92% reduction in read latency using this device mapper target.

Example dmsetup usage

unstriped ontop of Intel NVMe device that has 2 cores

```
dmsetup create nvmset0 --table '0 512 unstriped 2 256 0 /dev/nvme0n1 0'
dmsetup create nvmset1 --table '0 512 unstriped 2 256 1 /dev/nvme0n1 0'
```

There will now be two devices that expose Intel NVMe core 0 and 1 respectively:

```
/dev/mapper/nvmset0
/dev/mapper/nvmset1
```

unstriped ontop of striped with 4 drives using 128K chunk size

```
dmsetup create raid_disk0 --table '0 512 unstriped 4 256 0 /dev/mapper/striped 0'
dmsetup create raid_disk1 --table '0 512 unstriped 4 256 1 /dev/mapper/striped 0'
dmsetup create raid_disk2 --table '0 512 unstriped 4 256 2 /dev/mapper/striped 0'
dmsetup create raid_disk3 --table '0 512 unstriped 4 256 3 /dev/mapper/striped 0'
```