What is a Block Storage Device?

- A device that can read and write data in “blocks” of a given size (classically, 512 bytes, or a multiple of 512 bytes).

Examples of block devices:

- Hard disk drive
- Solid state drive (flash memory); NAND
- Tape drive
What is a Block Storage “Test” Device?

- A device that exhibits behavior simulating a certain failure mode, performance problem, or other condition, in order to test the behavior of applications using a block device.

Examples of block storage test devices:

- Linux kernel: scsi_debug (simulates SCSI disk behavior)
- Device-mapper: dm-flakey (simulates periodic I/O errors)
- Device-mapper: dm-delay (simulates excessive latency)
EXAMPLE TEST DEVICE
A device that sets the REQ_FUA bit on write bios at a given frequency, to test the behavior of devices undergoing excessive Force Unit Access writes.

```c
if (bio_data_dir(bio) == WRITE) {
    if ((fd->frequency != 0)
        && (atomic_inc_return(&fd->counter) % fd->frequency == 0)) {
        bio->bi_opf |= REQ_FUA;
    } else {
        bio->bi_opf |= ~REQ_FUA;
    }
}
return DM_MAPIO_REMAPPED;
```
EXAMPLE TEST DEVICE

A “forgetful” device, which honors flushing, but can simulate lost writes after a simulated device reset. Torn writes can be simulated via a mask.

torn_mask: 3043603817 (0b10110101011010011011010101101001)
torn_modulus: 16

0 DIRTY 0 0
3 DIRTY 0 3
5 DIRTY 0 5
6 DIRTY 0 6
8 DIRTY 0 8
10 DIRTY 0 10
12 DIRTY 0 12
13 DIRTY 0 13
EXAMPLE TEST DEVICE

A “tracer” device, that tracks the checksum of sectors as they are read or written during a blktrace run.

```
blkparse -t -i dm-0.blktrace.0 | grep \ 408\ \+
0.000675631 17378  Q  WS 408 + 8 [dd]
0.000678751   0  m   N W 408 + 1 [testtracer1], hash: 3af838120480c63b312b4169ed6165e0
0.013761184   0  C  WS 408 + 8 [0]
0.154222614 17380  Q   R 408 + 8 [systemd-udevd]
0.156901300   0  m   N R 408 + 1 [testtracer1], hash: 3af838120480c63b312b4169ed6165e0
0.156910103   0  C   R 408 + 8 [0]
5.435392156 17384  Q  WS 408 + 8 [fio]
5.435395879   0  m   N W 408 + 1 [testtracer1], hash: 0ed4be323ef27b33b6cc600ef11d4ea0
5.436701069   0  C  WS 408 + 8 [0]
6.826483011 17383  Q   R 408 + 8 [systemd-udevd]
6.829287241   0  m   N R 408 + 1 [testtracer1], hash: 0ed4be323ef27b33b6cc600ef11d4ea0
6.829296214   0  C   R 408 + 8 [0]
```
TEST DEVICE INSPIRATION

During a test of a VDO volume, a bad sector developed on a drive, which triggered a failure:

```
kernl: sd 3:0:0:0: [sdc] FAILED Result: hostbyte=DID_OK
driverbyte=DRIVER_SENSE
kernl: sd 3:0:0:0: [sdc] Sense Key : Medium Error [current]
[descriptor]
kernl: sd 3:0:0:0: [sdc] Add. Sense: Unrecovered read error - auto reallocate failed
kernl: sd 3:0:0:0: [sdc] CDB: Read(10) 28 00 00 47 45 f0 00 00 08 00
kernl: blk_update_request: I/O error, dev sdc, sector 4670960
kernl: ata3: EH complete
kernl: uds: kvdo0:dedupeQ: readBufferedData got readFromRegion error: Input/output error (5)
```
EXISTING TOOLS

The bug could be reproduced with scsi_debug with “opts=0x2”, which fails reads on sector 0x1234, for 10 blocks.

However...

- OPT_MEDIUM_ERR_ADDR (Sector 0x1234) and OPT_MEDIUM_ERR_NUM (10) are **hardcoded** in the source of scsi_debug, and **cannot be changed without recompiling the Linux kernel**.

- The backing storage is based on RAM. Creating an 8 GB scsi_debug device consumes 8 GB of RAM. **This data will not persist after a system crash.**

- The scsi_debug “medium error” flag provides **no control of when to start failing reads**. The options are kernel command line options, so the device either always fails reads, or never fails reads.
NEW TEST DEVICE: DM-DUST

dm-dust: A device to simulate the behavior of bad sectors

Simulate the general behavior of a standard block device (analogous to a dm-linear device). Allow initial metadata and pre-staging to be written prior to the test, then...

- Choose arbitrary blocks to fail
- Enable failure at a point in time
- Persist data across reboots
- Emulate “remapped sector” after writes
Using the base code from an existing test target, fail reads on the 42\textsuperscript{nd} 4096-byte block when failReadOnBadBlock is true:

```c
static int dustMap(struct dm_target *ti, struct bio *bio)
{
    DustDevice *dd = ti->private;
    bio->bi_bdev = dd->dev->bdev;
    setBioSector(bio, dm_target_offset(ti, getBioSector(bio)));

    sector_t testSector = 42 * 8;
    if (bio_data_dir(bio) == READ) {
        if ((dd->failReadOnBadBlock) && (testSector == getBioSector(bio))) {
            return DM_MAPI0_KILL;
        }
    }

    return DM_MAPI0_REMAPPEd;
}
```
BAD BLOCK LIST

The “bad block list” is a red-black tree, using the rbtree library in the Linux kernel. Bad blocks are added via “dmsetup message” commands:

```
$ sudo dmsetup message dust1 0 addbadblock 60
$ sudo dmsetup message dust1 0 removebadblock 60
```
ENABLING BAD BLOCK EMULATION
The device acts as a normal “pass-through” device until the bad block behavior is “enabled”:

```
$ sudo dmsetup message dust1 0 enable
kernel: device-mapper: dust: enabling read failures on bad sectors

$ sudo dmsetup status dust1
 0 33552384 dust 252:17 fail_read_on_bad_block
```
FAILING READS

Given a dust device with sector “67” in the bad block list, and “bad blocks” enabled, trying to read from sector 67 returns EIO (Input/output error):

```
$ sudo dd if=/dev/mapper/dust1 of=/dev/null bs=512 count=1 skip=67 iflag=direct
```
```
dd: error reading '/dev/mapper/dust1': Input/output error
0+0 records in
0+0 records out
0 bytes copied, 0.00040651 s, 0.0 kB/s
```
REMAPPING WRITES

Given a dust device with bad blocks added, and enabled, the device emulates sector remaps, allowing future reads to succeed.

```
$ sudo dmsetup message dust1 0 addbadblock 60
$ sudo dmsetup message dust1 0 addbadblock 67
$ sudo dmsetup message dust1 0 addbadblock 72
$ sudo dmsetup message dust1 0 addbadblock 87
$ sudo dmsetup message dust1 0 enable

$ sudo dd if=/dev/zero of=/dev/mapper/dust1 bs=512 count=128 oflag=direct
128+0 records in
128+0 records out

kernel: device-mapper: dust: block 60 removed from badblocklist by write
kernel: device-mapper: dust: block 67 removed from badblocklist by write
kernel: device-mapper: dust: block 72 removed from badblocklist by write
kernel: device-mapper: dust: block 87 removed from badblocklist by write
```
RELEASE TO UPSTREAM

The dm-dust device has been submitted to the upstream device-mapper mailing list, to (hopefully) be included in the Linux kernel.

Mailing list URL: https://www.redhat.com/mailman/listinfo/dm-devel

Remaining work for dm-dust:

- Block device limit stacking checks (physical_block_size)
- Variable “block size” option for simulated read failure
- Other changes required for upstream inclusion
QUESTIONS?
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHat