Red Hat

VM Block Error Injection, A Novel* Approach For Testing Linux Storage

Tony Asleson <tasleson@redhat.com>

January 16, 2020

* Well... **not** that novel

Data Is The New Bacon!

-My vegetarian friends t-shirt

“Data is a precious thing and will last longer than the systems themselves.”

– Tim Berners-Lee

Some Background

Block Storage Device

- Randomly accessible, fixed size blocks, eg. rotating magnetic media, solid state disk, etc.
- Typical size is 512, 4096 bytes
- What you can use to create a file system upon, however it’s not required for every file system
- Blocks are referenced by a logical block address, 0 ... N-1

Some Background

Block error for Small Computer System Interface (SCSI)

- OS/initiator issues command (read/write)
- Disk/target processes command returns status with/without data
- If status is error, OS/initiator requests additional information from the disk/target (request sense)
- Operating system decodes sense data and does reporting and potentially recovery

Storage Device Errors

Errors that operating systems need to handle

- Temporary or persistent read/write error
- Read error corrected by write
- Temporary or persistent timeouts
  - Why can this be worse than a hard error?
- Unexpected resets (device spontaneously restarts)
- Detect incorrect data, report error and/or correct
- High latency/poor performance, possibly in the presence of errors

Importance Of Testing The Storage Stack

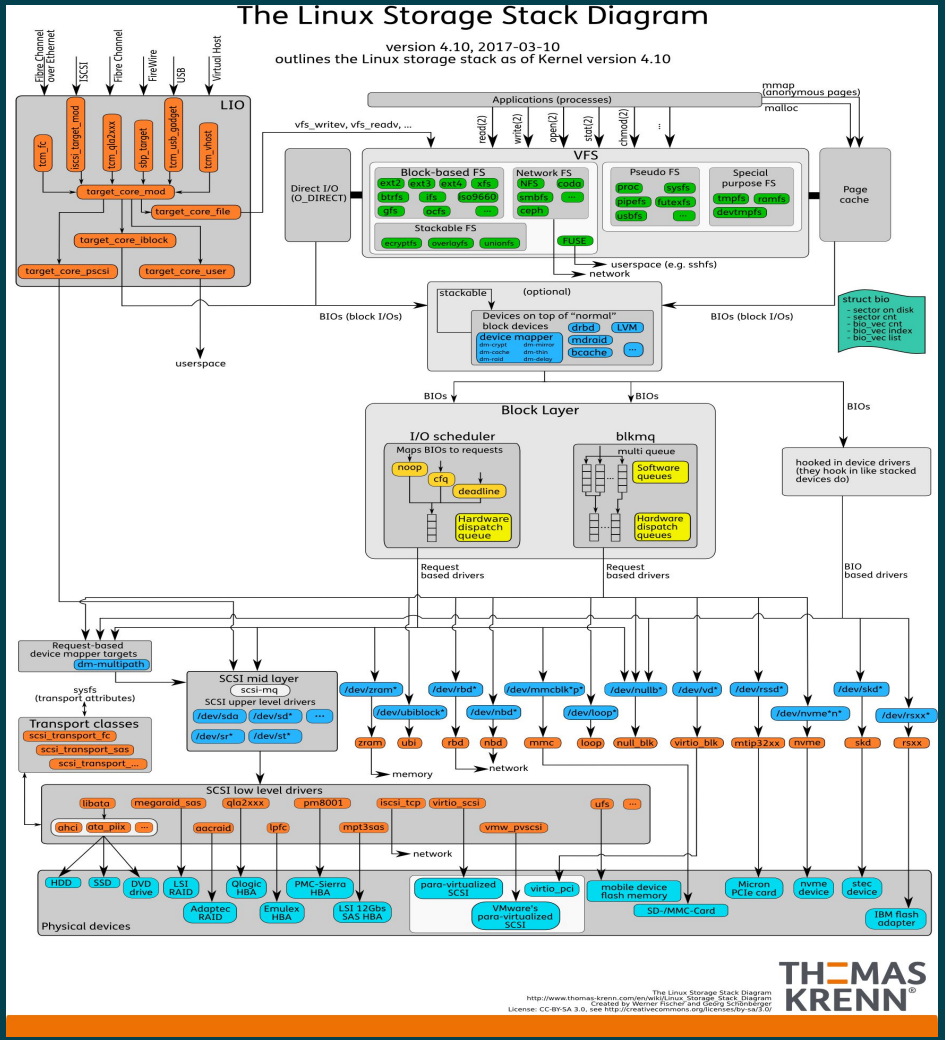
- Operating systems require the ability to gracefully handle storage hardware errors
  - No one wants their system to crash if a storage error occurs
- Need to exercise error code paths to ensure
  - Data integrity
  - Adequate logging
  - No kernel panics (non-intentional anyway)
  - No memory leaks
  - Correct recovery behavior (retry, reset, RAID correction)

Layering Of Linux Storage Stack

- Where you generate the error determines which layers of the stack get tested
- What does the internal architecture of Linux storage stack look like?

Linux Storage Stack

It's complicated



Red Hat

Difficulties In Testing

Storage devices are quite reliable

- Advertised rate by an enterprise Serial Attached SCSI disk drive (SAS)
  - Un-recovered Less than 1 sector in 10^16 bits transferred
    - (10**16 bits) / 8 to bytes / (2 ** 40) to TiB = 1136 TiB
  - Miscorrected Less than 1 sector in 10^21 bits transferred
- Actual error rates can be worse, search CERN Data integrity, 10^7
- How can you effectively test error paths for events that rarely happen?

Characteristics Of Desirable Error Testing

- Availability
- Ease of use
- Repeatability
- Low monetary cost
- Simple application programming interface (API) for automation

Different Approaches In Creating Errors

- “Fake” errors
  - In kernel device(s)
  - External network storage device
- “Actual” errors (emanate from hardware)
  - Use actual hardware
  - Virtual machine (simulated hardware)

In Kernel Device And Network

Not a complete list

- Write an in kernel device or layering device which creates the needed errors
  - SCSI debug
  - dm-flakey, dm-delay, dm-dust
  - SCSI Fault injector
- Use network device which return errors
  - Network block device (NBD)

SCSI Debug

- Simulates 1 or more SCSI devices
- RAM backed, not persistent, limited to available system memory
- Has runtime options in sysfs for configuration
  - Medium_error_start, medium_error_count, timeouts, delays, recovered media error, aborted commands, device queue full ...

Device Mapper Error Targets

- Device mapper (dm) targets can be layered over other dm devices or actual storage devices
- Dm-flaky - Starting from the time the table is loaded, the device is available for N seconds, then exhibits unreliable behaviour for N seconds, and then the cycle repeats
- Dm-delay - A target that delays reads and/or writes and can send them to different devices
- Dm-dust - Generate read errors and read errors that can be corrected with a write

Network Block Device (NBD)

- Can create block devices from files or in memory
  - Sparse support, so you can create sizes that exceed actual hardware limits, eg. 8EiB
- Errors can be created for a block device by creating a file in /tmp
  - For /dev/nbd0 -> touch /tmp/error0 (error file is configurable)
- Can create read delays, write delays
- Can set error rates as a percentage or probability

SCSI Fault Injector

- Combination of SystemTap for kernel instrumentation and the external program SCSI fault injector which maintains state and dictates actions
- Created in the kernel 2.6 time frame, circa 2008
- Seems like maintenance and updates have stalled
- It found a number of different bugs during development
- https://www.kernel.org/doc/ols/2008/ols2008v2-pages-205-214.pdf

Use An Actual SCSI Disk Drive

Warning: Don’t do this on a drive you value

- Read long -> corrupt bytes in buffer -> write long -> regular read = read error
- Mode page settings to discover size of correction span
- Fix by rewriting with regular write
- Can prematurely age drive due to increased error counts
- Can cause auto re-allocates which may fill the grown defect list and possibly cause drive to fail
- May cause SMART errors which may be a good thing for testing
- What disk drive devs do, they have own functionality to clear drive
- Errors limited to read errors (recoverable and unrecoverable)

Create The Errors In A Virtual Machine (VM)

- If we are already simulating the hardware, why not simulate possible error responses too?
- Seems like a great way to ensure correct behavior of guest operating systems

Benefits Of Adding Error Injection In VM

- Can present errors before the OS or even the boot loader gets loaded
  - Ensure your RAID solution actually allows you to boot in a degraded mode
- Operating system agnostic, you can test any OS that will run in VM
  - Compare/contrast file system implementations, volume managers, software RAID
- Exercise more layers of the storage stack
  - Note: Limited to hardware emulation, thus not all storage device drivers can be tested
- No resources consumed from guest OS, does consume host resources

Benefits Of Adding Error Injection In VM

Continued

- No artificial test code in kernel, test like you would in production
- Anyone that can run the VM environment can use, no special hardware
- Create errors for all the supported device types and attachment options
  - SCSI (Parallel, SAS, FC)
  - ATA (PATA, SATA)
  - NVMe
  - Others ...

Risks Of VM Error Injection

What could possibly go wrong

- Because we are mimicking hardware we need to make sure that it adheres to the interface protocol
  - Want to avoid programing to incorrect behavior
  - This happens with real hardware, vendors incorrectly implementing a protocol, kernel has device specific code to handle this
- VMs accurately reflect the hardware implementation they are trying to model, even the issues, they have to model hardware bugs too

Other Potential Use Cases

Some of These Already Exist

- Statistics gathering (blktrace for all)
  - Transfer size
  - Location / hot spots
  - Access patterns
- Capture / Playback
  - Capture sequence, play it back
- Repeatability for error reproduction or analysis

Future Ideas

- Create shingled magnetic recording (SMR) device
  - Allow developers to create new device mapper or filesystems to improve usability and performance
- Expand device models to support more features

Proof Of Concept For QEMU

- https://github.com/tasleson/qemu/tree/block_error_inject
- Adds a QAPI for adding/removing media errors for 1 or more block devices
- Modifications to SCSI, AHCI, NVMe block devices
  - Ability to identify which logical block is in error for request
  - Returns accurate error data, SCSI sense data with sector in error

Proof Of Concept For QEMU

Continued

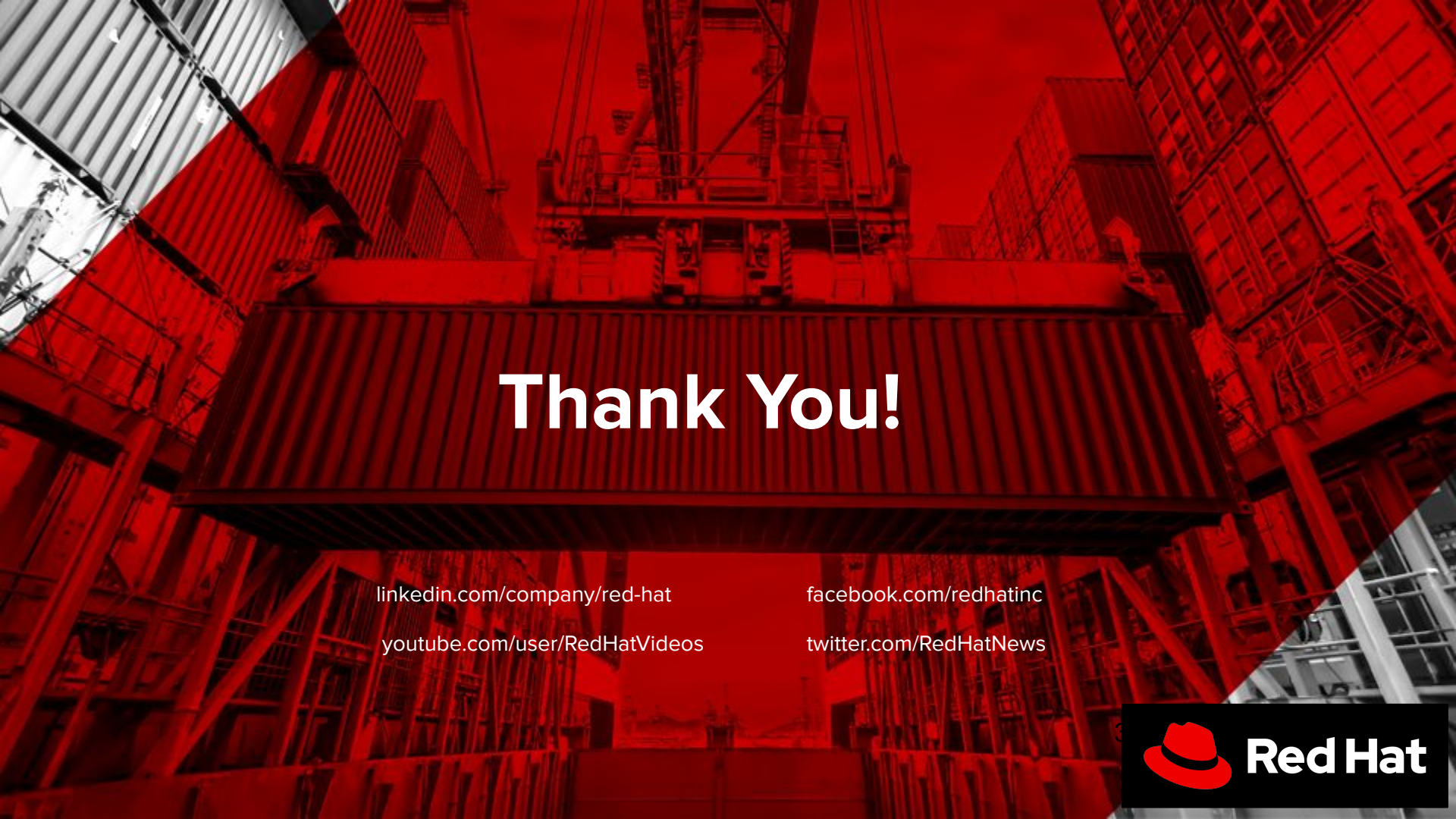
- QEMU already has the ability to inject some errors
  - blkdebug
    - Utilizes a configuration file
    - Can have logic based on different sequence of events
  - I wasn't aware of this when I added my functionality
- Plan is to merge the functionality I added with the existing blkdebug and extend the features

An Example Use

- I proposed a logging change to Linux kernel to add a unique durable ID to storage related messages
- It required forcing the kernel down storage error paths to test logging changes for correctness
- Having this functionality in QEMU made this process much easier, especially considering kernel changes made in different storage subsystems

What Else Can We Test With A VM?

Questions?


Thank You!
linkedin.com/company/red-hat
facebook.com/redhatinc
youtube.com/user/RedHatVideos
twitter.com/RedHatNews
Red Hat