A Generic Framework for Testing Parallel File Systems

Jinrui Cao†, Simeng Wang†, Dong Dai‡, Mai Zheng†, and Yong Chen‡
† Computer Science Department, New Mexico State University
‡ Computer Science Department, Texas Tech University

Presented by Simeng Wang
SC16' PDSW-DISCS
11. 14. 2016.

Motivation

Subject: Update: HPCC Power Outage
Date: Monday, January 11, 2016 at 8:50:17 AM Central Standard Time
From: HPCC - Support
Attachments: image001.png, image003.png

TEXAS TECH UNIVERSITY
Information Technology Division

High Performance Computing Center

Jan, 2016 @HPCC: power outage lead to unmeasurable data loss

To All HPCC Customers and Partners,

As we have informed you earlier, the Experimental Sciences Building experienced a major power outage Sunday, Jan. 3 and another set of outages Tuesday, Jan. 5 that occurred while file systems were being recovered from the first outage. As a result, there were major losses of important parts of the file systems for the work, scratch and certain experimental group special Lustre areas.

The HPCC staff have been working continuously since these events on recovery procedures to try to restore as much as possible of the affected file systems. These procedures are extremely time-consuming, taking days to complete in some cases. Although about a third of the affected file systems have been recovered, work continues on this effort and no time estimate is possible at present.

Motivation

- Existing methods for testing storage systems are not good enough for large-scale parallel file systems (PFS)
  - Model checking [e.g., EXPLODE@OSDI'06]
    - difficult to build a controllable model for PFS
    - state explosion problem
  - Formal methods [e.g., FSCQ@SOSP'15]
    - challenging to write correct specifications for PFS
  - Automatic Testing [e.g., TorturingDB, CrashConsistency@OSDI'14]
    - closely tied to local storage stack: intrusive for PFS
    - only work for single-node

Our Contributions

- A generic framework for testing failure handling of parallel file system
  - Minimal interference & high portability
    - decouple PFS from the testing framework through a remote storage protocol (iSCSI)
  - Systematically generate failure events with high fidelity
    - fine-grained, controllable failure emulation
    - emulate realistic failure modes

- An initial prototype for **Lustre** file system
  - Uncover internal I/O behaviors of Lustre under different workloads and failure conditions

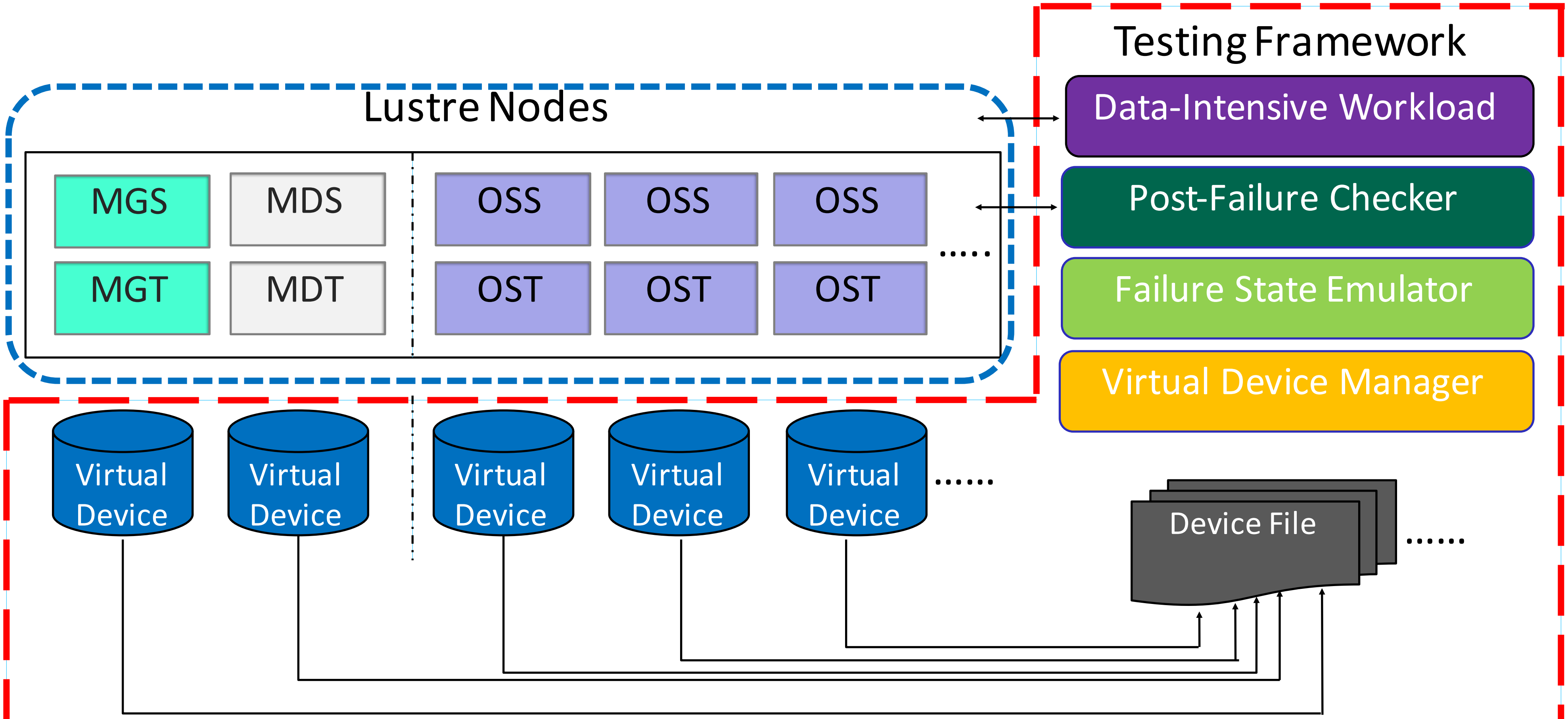
Outline

- Introduction
- Design
  - Virtual Device Manager
  - Failure State Emulator
  - Data-Intensive Workloads
  - Post-Failure Checker
- Preliminary Experiments
- Conclusion and Future Work

Outline

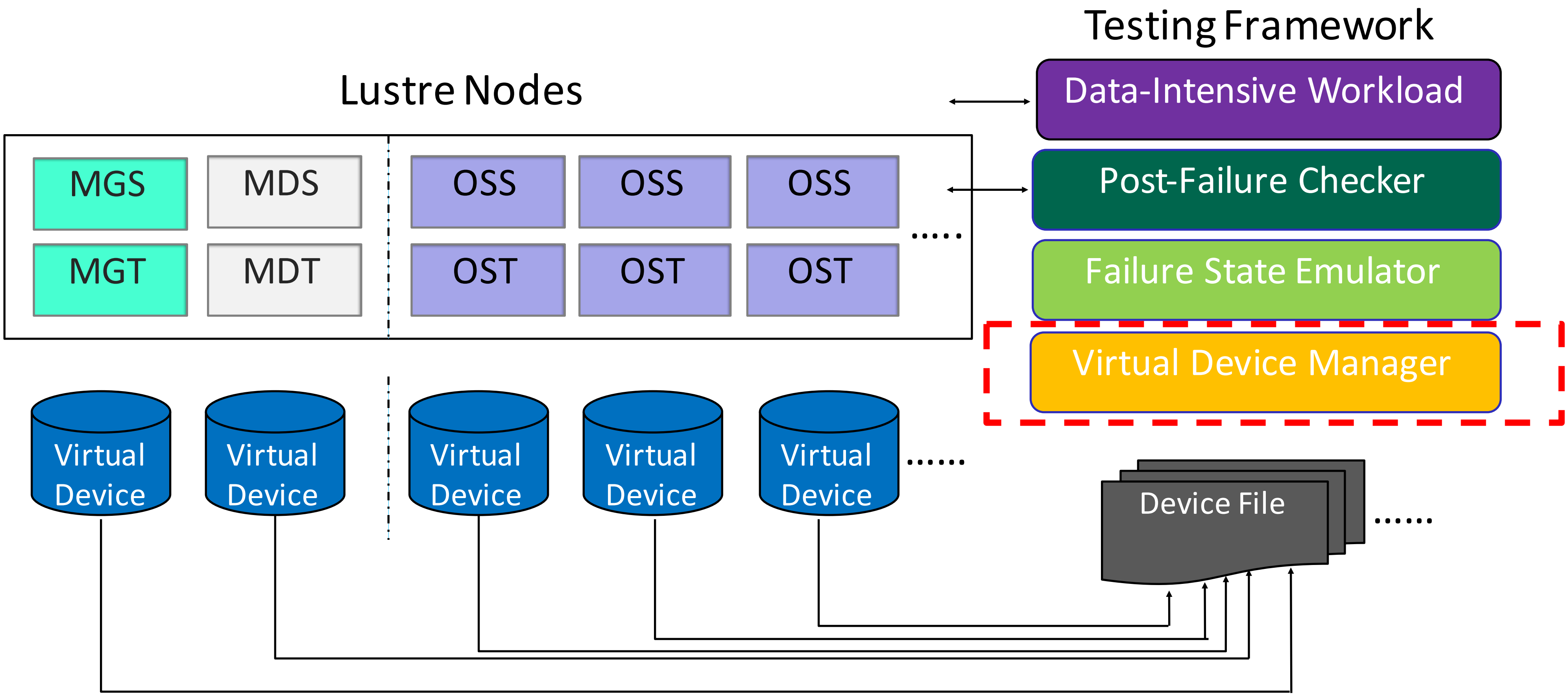
- Introduction
- Design
  - Virtual Device Manager
  - Failure State Emulator
  - Data-Intensive Workloads
  - Post-Failure Checker
- Preliminary Experiments
- Conclusion and Future Work

Overview



MGS: Management Server MGT: Management Target
MDS: Metadata Server MDT: Metadata Target
OSS: Object Storage Server OST: Object Storage Target

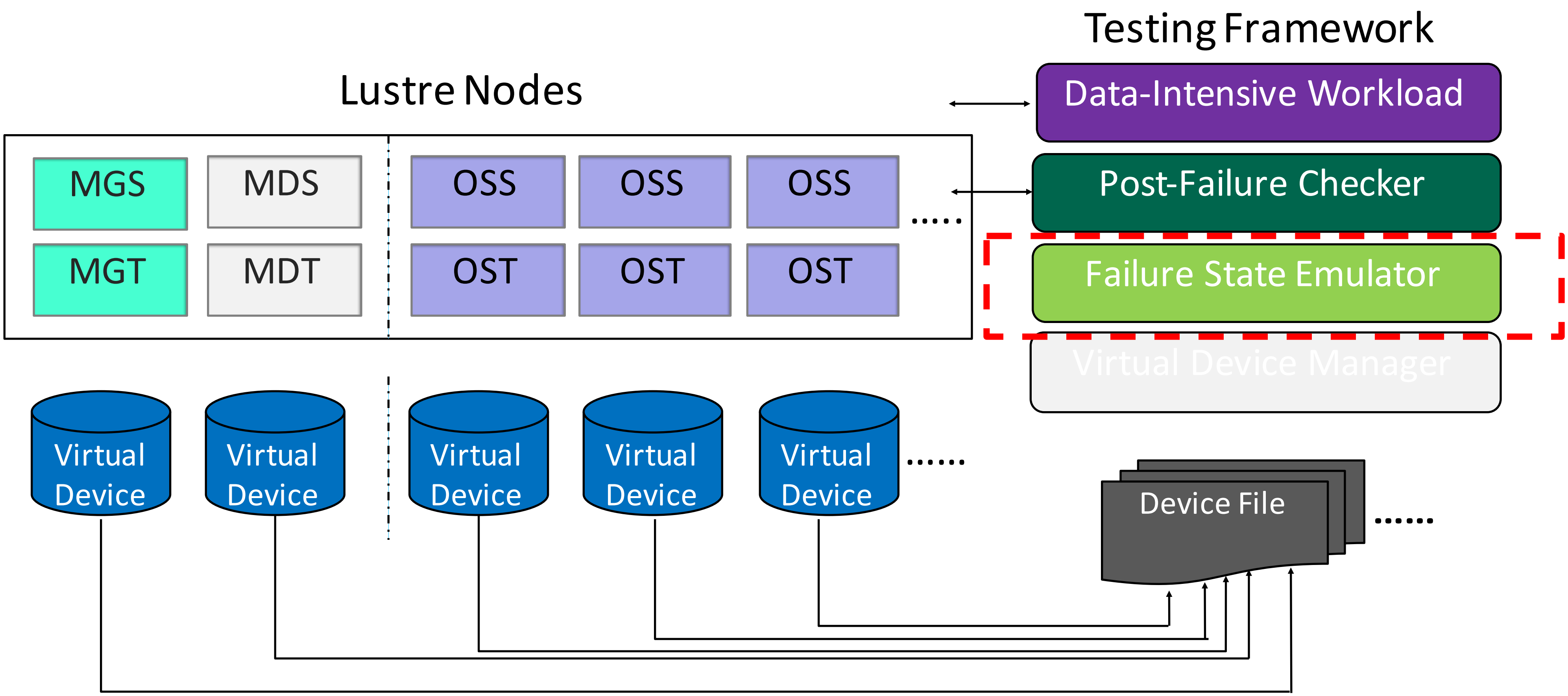
Overview


Testing Framework
Data-Intensive Workload
Post-Failure Checker
Failure State Emulator
Virtual Device Manager
Lustre Nodes
MGS
MDS
OSS
OSS
OSS
MGT
MDT
OST
OST
OST
Virtual Device
Virtual Device
Virtual Device
Virtual Device
Virtual Device
Device File

Virtual Device Manager

- Creates and maintains device files for storage devices.
- Mounted to Lustre nodes as virtual devices via iSCSI.
- I/O operations are translated into disk I/O commands
    - Log commands into a command history log
    - Include node IDs, command details, and actual data transferred
    - Used by the **Failure State Emulator**

Overview



Testing Framework
Lustre Nodes
Data-Intensive Workload
Post-Failure Checker
Failure State Emulator
Virtual Device Manager
MGS
MDS
OSS
OSS
OSS
MGT
MDT
OST
OST
OST
.....
Virtual Device
Virtual Device
Virtual Device
Virtual Device
Virtual Device
......
Device File
......

Failure State Emulator

- Generate failure events in a **systematic** and **controllable** way.
  - Manipulate I/O commands and emulates failure state of each individual device
  - Emulate four realistic **failure modes** based on previous studies [e.g., FAST'13, OSDI'14, TOCS'16, FAST'16]

1. Whole Device Failure

   Device becomes invisible to the host

2. Clean Termination of Writes

   Emulates simplest power outage

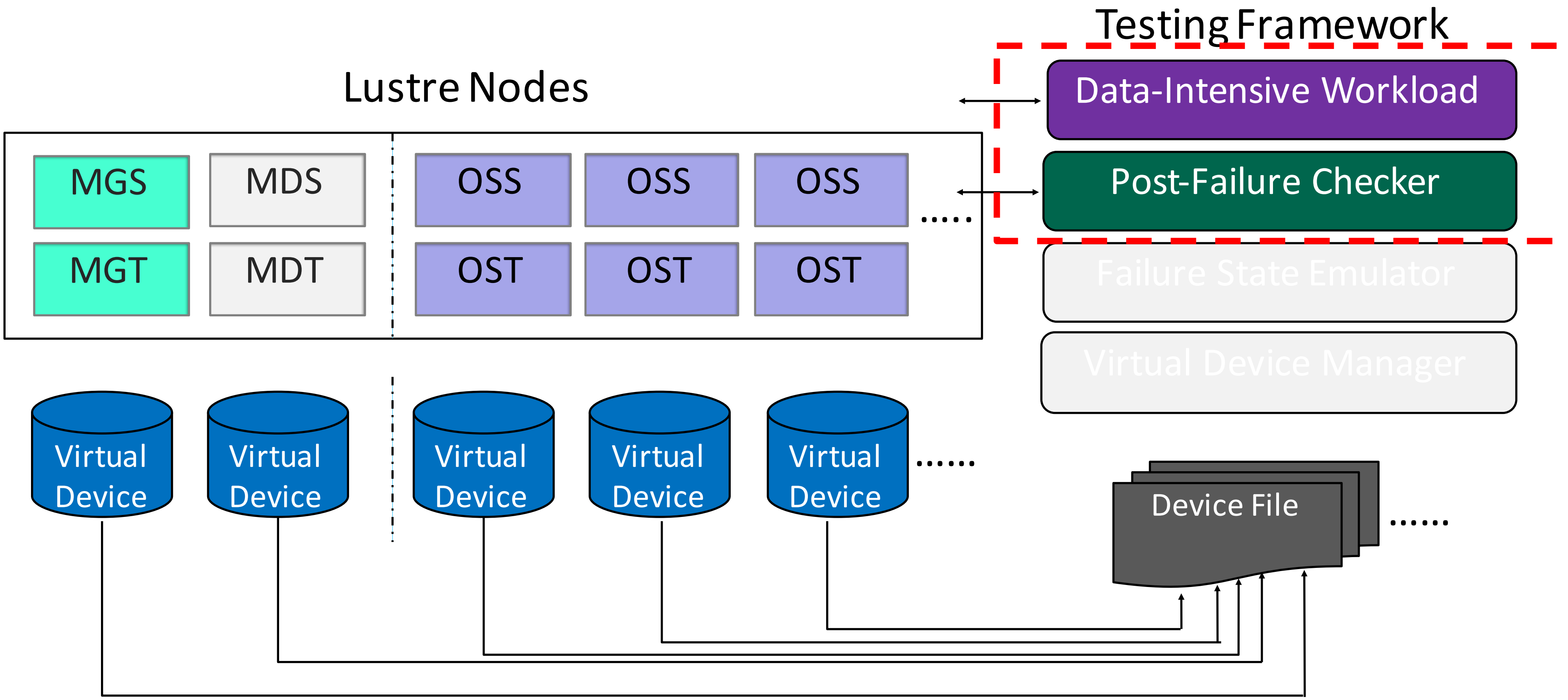
3. Reordering of the Writes

   Commits writes in an order different from the issuing order

4. Corruption of the Device Block

   Change content of writes

Overview


Testing Framework
Lustre Nodes
Data-Intensive Workload
Post-Failure Checker
Failure State Emulator
Virtual Device Manager
MGS
MDS
OSS
OSS
OSS
MGT
MDT
OST
OST
OST
.....
Virtual Device
Virtual Device
Virtual Device
Virtual Device
Virtual Device
......
Device File
......

Co-design Workloads and Checkers

- Data-Intensive workloads
  - Stress **Lustre** and generate I/O operations to age the system and bring it to a state that may be difficult to recover
  - May use existing data-intensive workloads
  - May include self-identification/verification information

- Post-Failure Checkers
  - examines the post-failure behavior and check if it can recover without data loss
  - May use existing checkers (e.g.,, LFSCK for Lustre)

Outline

- Introduction
- Design
  - Virtual Device Manager
  - Failure State Emulator
  - Data-Intensive Workloads
  - Post-Failure Checker
- Preliminary Experiments
- Conclusion and Future Work

Preliminary Experiment

- Experiment setup
  - Cluster of **seven** VMs, installed with CentOS 7.
  - Lustre file system (version 2.8) on five VMs.
  - **One** MGS/MGT node, **one** MDS/MDT node, and **three** OSS/OST nodes.
  - **Sixth VM**: hosts the **Virtual Device Manager** and the **Failure State Emulator**
    - **Virtual Device Manager** is built on top of the Linux SCSI target framework
  - **Last VM**: used as client for launching workloads and LFSCK
    - **Data-Intensive Workload**, **Post-Failure Checker**

Preliminary Experiment

- Workloads
  - Normal Workloads ran on Lustre

Workload	Description
Montage/m101	astronomical image mosaic engine
cp	copy a file into Lustre
tar	decompress a file on Lustre
rm	delete a file from Lustre

  - Post-Failure Workloads ran on Lustre

Operation	Description
lfs setstripe	set striping pattern
dd-nosync	create & extend a Lustre file
dd-sync	create & extend a Lustre file
LFSCK	check & repair Lustre

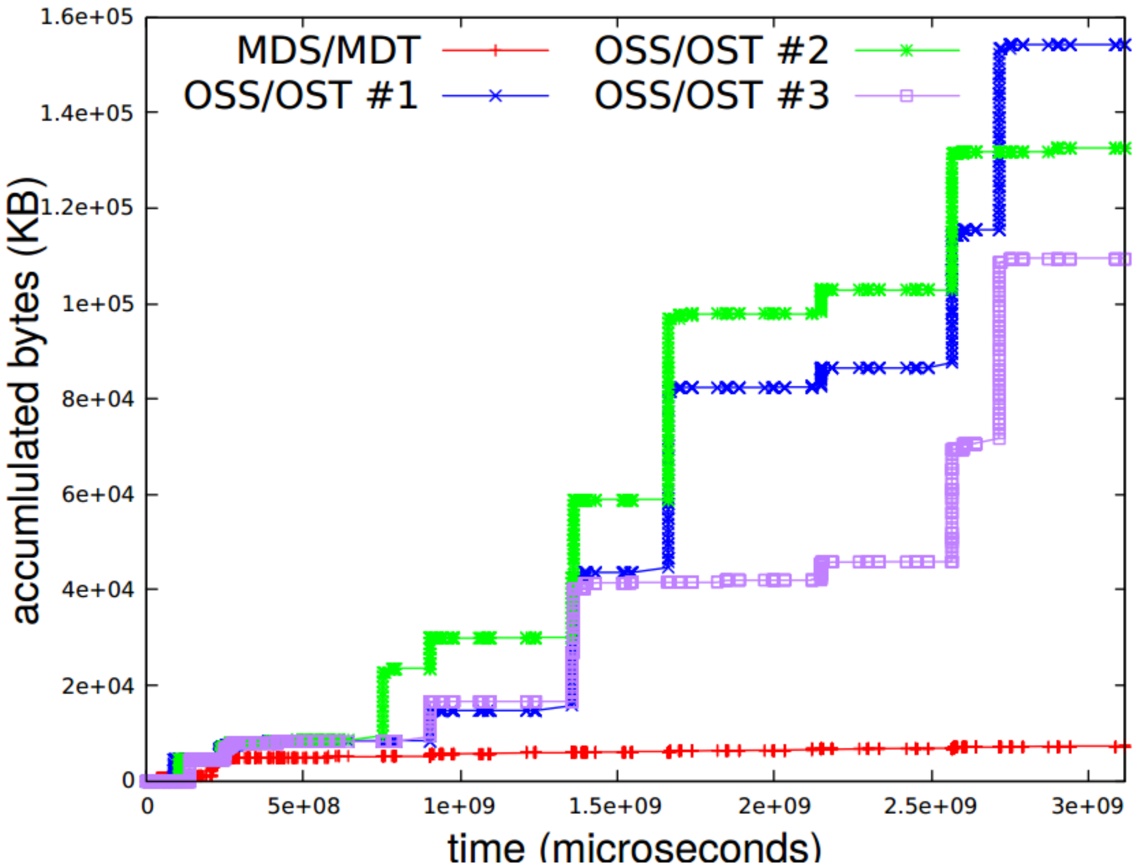
Preliminary Results

- Internal Pattern of Writes without Failure
  - Numbers of bytes (MB) written to different Lustre nodes under different workloads.
  - Montage/m101 is spilt into twelve steps (i.e., s1 to s12) to show the fine-grained write pattern.

Luster Nodes	cp	tar	rm	Montage/m101											
				s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12
MGS/MGT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MDS/MDT	0.1	5	0.2	6	0.4	6	0.5	6	0.6	6	0.7	6	1	6	1
OSS/OST#1	0	14	0	14	28	14	66	14	66	18	66	18	94	56	94
OSS/OST#2	15	14	15	14	43	14	81	14	81	19	81	19	109	19	110
OSS/OST#3	0	16	0	16	24	16	24	17	24	21	24	21	49	58	49

Preliminary Results

- Internal Pattern of Writes without Failure
  - Accumulated numbers of bytes (KB) written to different nodes during the workloads.



Preliminary Results

- Post-Failure Behavior
  - Emulate a whole device failure on MDS/MDT node
  - Run operations on Lustre after the emulated device failure
    - dd-nosync means using dd to create and extend a Lustre file
    - dd-sync means enforcing synchronous writes on the dd command
    - The last column shows whether the operation reported error or not

Operation	Description	Report Error?
lfs setstripe	set striping pattern	No
dd-nosync	create & extend a Lustre file	No
dd-sync	create & extend a Lustre file	Yes
LFSCK	check & repair Lustre	No

Outline

- Introduction
- Design
  - Virtual Device Manager
  - Failure State Emulator
  - Data-Intensive Workloads
  - Post-Failure Checker
- Preliminary Experiments
- Conclusion and Future Work

Conclusion and Future Work

- Proposed and prototyped a framework for testing failure handling of large-scale parallel file systems.
- Uncovered internal behaviors towards workloads under normal and failure conditions
- More effective post-failure checking operations
- More file systems (e.g., PVFS, Ceph)
- Explore novel mechanisms to enhance the resilience of large-scale parallel file systems

Thank You!

Questions ?