A Performance Study of LFSCK

Bottlenecks and Potentials

Dong Dai[1], Om Rameshwar Gatla[2], Mai Zheng[2]

[1]University of North Carolina at Charlotte
[2]Iowa State University

UNC CHARLOTTE
College of Computing and Informatics

STATE

Outline

- Motivation
- Background
- Performance Study Methods
- Results and Analysis
- Potentials
- Conclusions and Plans

Motivation

- Lustre is one of the mostly used high performance parallel file systems on the market
- It is deployed on large and complex systems
  - ANL's coming Exascale machine: Aurora[1]
  - Planned to use Lustre file system
  - File System Capacity: > 150 petabytes
  - File System Throughput: > 1 terabyte/s
- Complex systems may fail for various of reasons
  - Management issues
  - Unknown software/hardware bugs
- Extreme failure cases may lead to data loss
  - LFSCK

[1] ANL Aurora Programs: https://www.alcf.anl.gov/programs/aurora-esp

LFSCK

- The tool to check and fix problems of Lustre file system
  - lfsck: early version of file system checker for Lustre
    - Utilizes databases created via e2fsck (e2fsprogs)
    - Very slow due to large database size
  - LFSCK: online Lustre file system checker
    - Available in Lustre 2.x
    - Lustre 2.4: can verify and repair the directory FID-in-dirent and LinkEA consistency
    - Lustre 2.6: can verify and repair MDT-OST file layout consistency
    - Lustre 2.7: can support verify and repair inconsistencies between multiple MDTs
    - ...
- Current Status
  - LFSCK 1 -> LFSCK 1.5 -> LFSCK 2 -> LFSCK 3 -> LFSCK 4
  - LFSCK-4: LFSCK performance enhancement
    - LU-5820: evaluation: linkEA verification history in RAM performance

LFSCK - Issues

- LFSCK is a powerful tool to combat failures and inconsistencies in Lustre file system
- However, it is not perfect yet...
- Functionality
  - It might not be able to identify nor fix failures of Lustre
  - Please check our study in [ICS'18]

Node(s) Affected	Fault Models	LFSCK	WikiR	WikiW-async	WikiW-sync
MGS	a-DevFail	normal	✓	✓	✓
	b-Inconsist	normal	✓	✓	✓
	c-Network	normal	✓	✓	✓
MDS	a-DevFail	**Invalid**	hang	hang	hang
	a-DevFail (v2.10)	**I/O err**	I/O err	I/O err	I/O err
	b-Inconsist	normal	✓	✓	✓
	c-Network	**I/O err**	hang	hang	hang
	c-Network (v2.10)	**hang**	hang	hang	hang
OSS#2	a-DevFail	**hang**	hang	hang	hang
	a-DevFail (v2.10)	normal	✓	✓	✓
	b-Inconsist	**reboot**	corrupt	hang	hang
	c-Network	**hang**	hang	hang	hang
three OSSes	a-DevFail	**hang**	hang	hang	hang
	a-DevFail (v2.10)	normal	✓	hang	hang
	b-Inconsist	**reboot**	corrupt	hang	hang
	c-Network	**hang**	hang	hang	hang
MDS + OSS#2	a-DevFail	**Invalid**	hang	hang	hang
	a-DevFail (v2.10)	**I/O err**	I/O err	I/O err	I/O err
	b-Inconsist	**reboot**	corrupt	hang	hang
	c-Network	**I/O err**	hang	hang	hang
	c-Network (v2.10)	**hang**	hang	hang	hang

Table 2: Response of LFSCK and post-LFSCK Workloads. *The first column shows where the faults are injected. The second column shows the fault models applied (v2.10 means applying to Lustre v2.10). "normal": LFSCK finishes normally; "reboot": at least one OSS node is forced to reboot; "Invalid": an "Invalid Argument" error; "I/O err": an "Input/Output error"; "hang": cannot finish within one hour; " ✓ ": complete w/o error; "corrupt": checksum mismatch. The bold font highlights the unexpected response of LFSCK.*

[ICS'18] PFault: A General Framework for Analyzing the Reliability of High-Performance Parallel File Systems. Jinrui Cao, Om Rameshwar Gatla, Mai Zheng, **Dong Dai**, Vidya Eswarappa, Yan Mu and Yong Chen. Accepted to appear in the proceedings of the 32nd ACM/SIGARCH International Conference on Supercomputing (**ICS'18**), 2018.

LFSCK - Issues

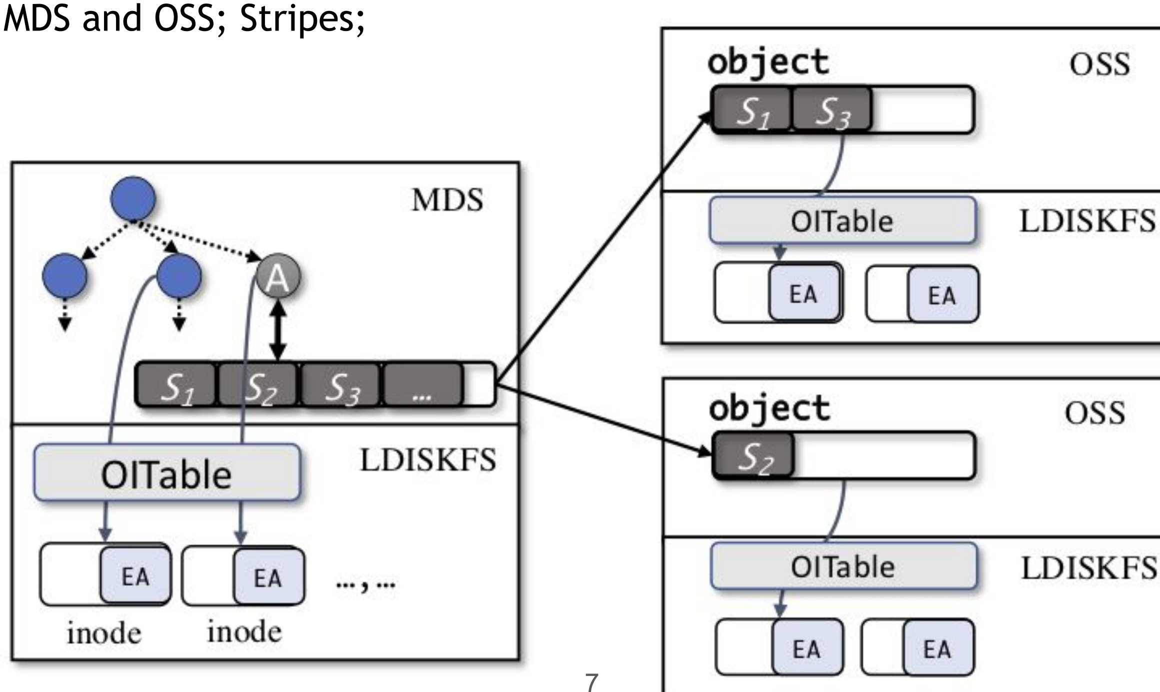
- LFSCK is a powerful tool to combat failures and inconsistencies in Lustre file system
- However, it is not perfect yet...
- Performance
  - Sometimes, we observed that running file system checking could be still slow, especially on a well-aged Lustre
  - The slowness might come from e2fsck or LFSCK itself. But, we focus on LFSCK in this study

> *"... experienced a major power outage Sunday, Jan. 3 and another set of outages Tuesday, Jan. 5 that* ***occured while file system were being recovered from the first outage****. As a result, there* ***were major losses*** *of important parts of the file systems for .... Lustre areas..."*
>
> \- *Quoted from the email about the failure*

UNC CHARLOTTE
College of Computing and Informatics

LFSCK Basis

- Lustre Metadata Management
  - **Mapping metadata between global entity and local inode**
    - MDS and OSS; Object Index; OITable
  - **POSIX namespace metadata**
    - MDS only; directory-tree structure
  - **Distributed data layout metadata**
    - MDS and OSS; Stripes;

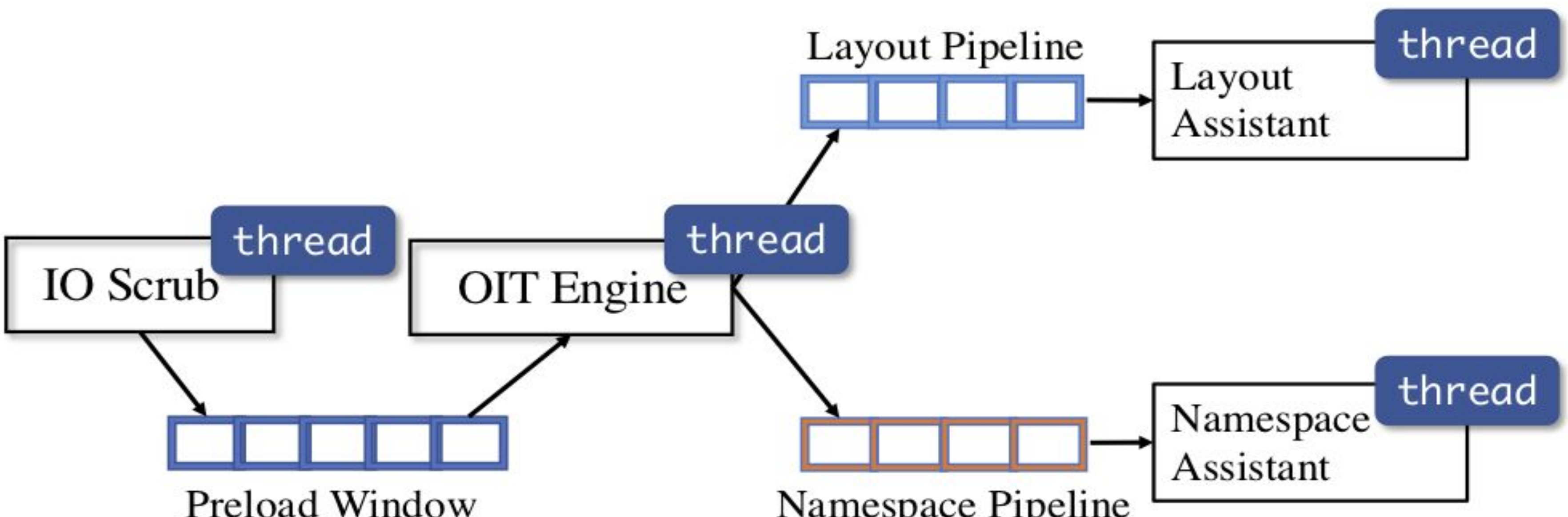


LFSCK Basis Cont’

- Lustre stores these metadata in more than one places
    a. So that LFSCK can cross-check them
    b. For example, global file -> local inode; local inode also has global fid
- LFSCK is designed to be a two-stage procedure
    a. MDS drives the OSS nodes to conduct checking and potential local fix
    b. Once finish, all OSS nodes will start the second stage to resolve orphan and missing objects detected in the first stage.
- We focus on the first stage
    a. as most of the time, the number of inconsistencies in a healthy Lustre should be minimal;
        - => indicating a short second stage
    b. while, we still need to run LFSCK oftenly to guarantee a healthy Lustre;
        - => indicating its importance regarding the performance

LFSCK Details Cont’

- LFSCK is implemented as several kernel threads connected with kernel buffers.
    - IOScrub sequentially scans the local disks
    - OIT Engine is the driving engine of LFSCK. It checks OITable
    - OIT Engine and IOScrub share buffer, sized `SCRUB_WINDOW_SIZE`
    - Layout/Namespace assistant threads will check Layout and Namespace metadata respectively
    - The buffer sizes are `LFSCK_ASYNC_WIN_DEFAULT`



Performance Study Methods

- Evaluation Platform
  - Our testbed is built on NSF Cloudlab platform
  - Small cluster with 1 MDS node, 8 OSS nodes, and 4 client nodes.
  - Each node is a c220g1 node
    - Two E5-2630 CPU
    - 128 GB Memory
    - 1.2 TB 10K RPM Hard disk
    - 480GB Intel DC3500 SSD
    - 10GB Intel NIC
  - CentOS 7.3
  - Lustre 2.10.4
  - Reproducible using the same Clouldlab profile
    - We will public our profile and all the test scripts

Performance Study Methods

- Aging Method
  - It is non-trivial to age a file system into proper state for performance testing on file system checkers
    - Especially true for large-scale parallel file system
  - We leveraged Darshan in this study
    - Darshan is an I/O characterization tool for HPC
    - Developed by ANL team
    - Deployed at various supercomputers
  - Public Darshan logs collected from Intrepid
    - Record all I/O activities done by each applications
  - Two Key I/O metadata
    - CP_SIZE_AT_OPEN: how large the file was when the application opened it
    - CP_BYTES_WRITTEN: how many bytes were written to the file by this application

DARSHAN
HPC I/O Characterization Tool

Performance Study Methods

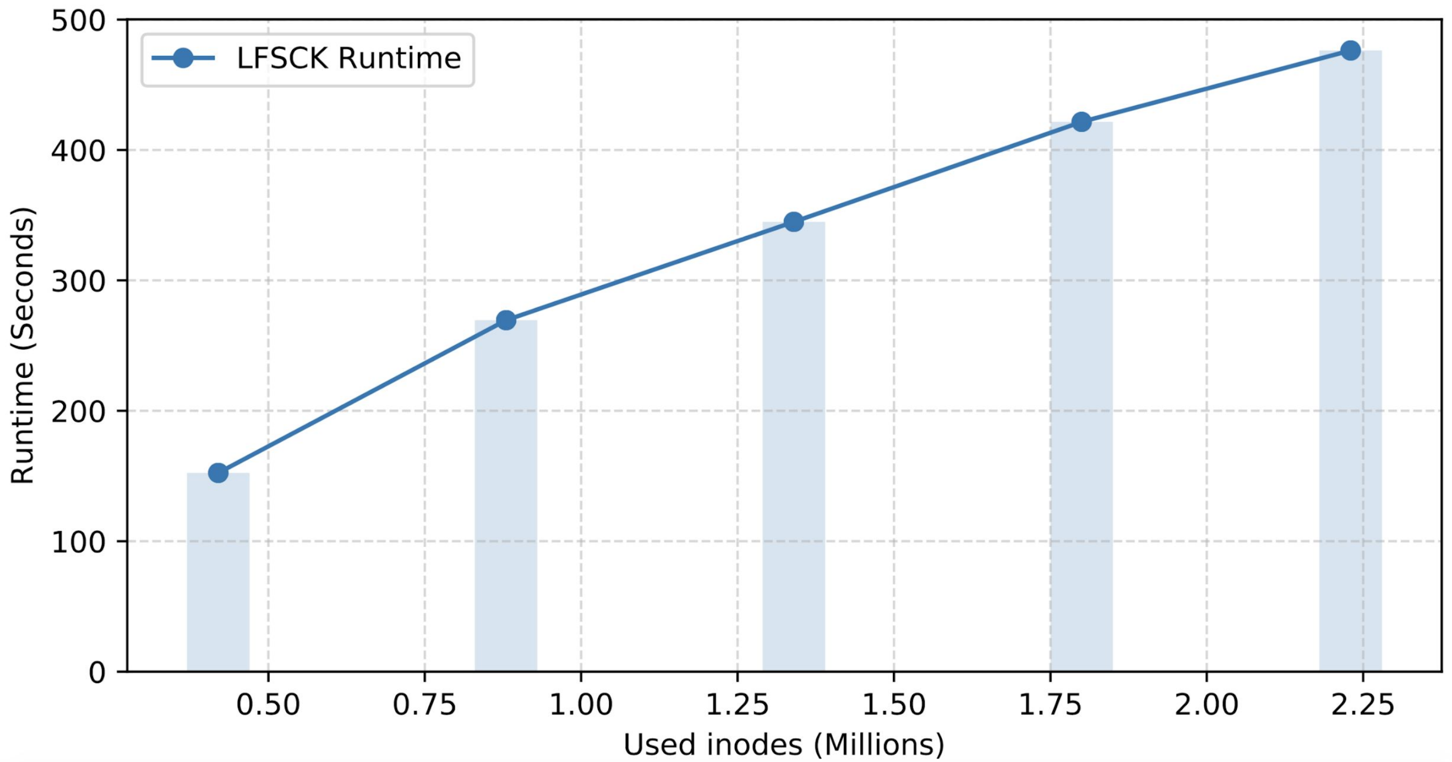
- Random paths
  - Darshan logs were anonymized. No full path or name of the file
  - Generate the structure manually, randomly
    - Randomly generate a directory with random (1->10) depth
    - Randomly generate files (1->100) into the same directory
- Reduce data sizes
  - Files in real-world supercomputers are really big
    - There are files that over 1TB each
  - Reduce files that are larger than 1MB * 8 to be 8MB.
    - Keep the metadata the same
    - Minimize the occupied data size
- Set Stripe Count
  - We set stripe count to be “-1” and stripe size to be “1MB”
  - Maximize the number of stripes
  - Note that, this setting is **not typical** in production system.

Performance Study Methods

- Monitoring Framework
  - The key is to monitor the execution of LFSCK
  - We leverage two tools
  - **<u>iostat</u>**
    - monitor the storage devices on all MDS and OSS nodes where Lustre was mounted.
    - `iostat -p sdc -d 1`
  - **<u>netstate</u>**
    - monitor the utilization of the network cards in the cluster used by Lustre
    - `netstat --interfaces=enp6s0f0 -c 1`

Results and Analysis

- Scalability
  - How LFSCK reacts to increasing number of files (inodes) in the system?
  - 8 OSS nodes and 1 MDS node
  - inode increases from 0.4 million to 2.3 million



Results and Analysis

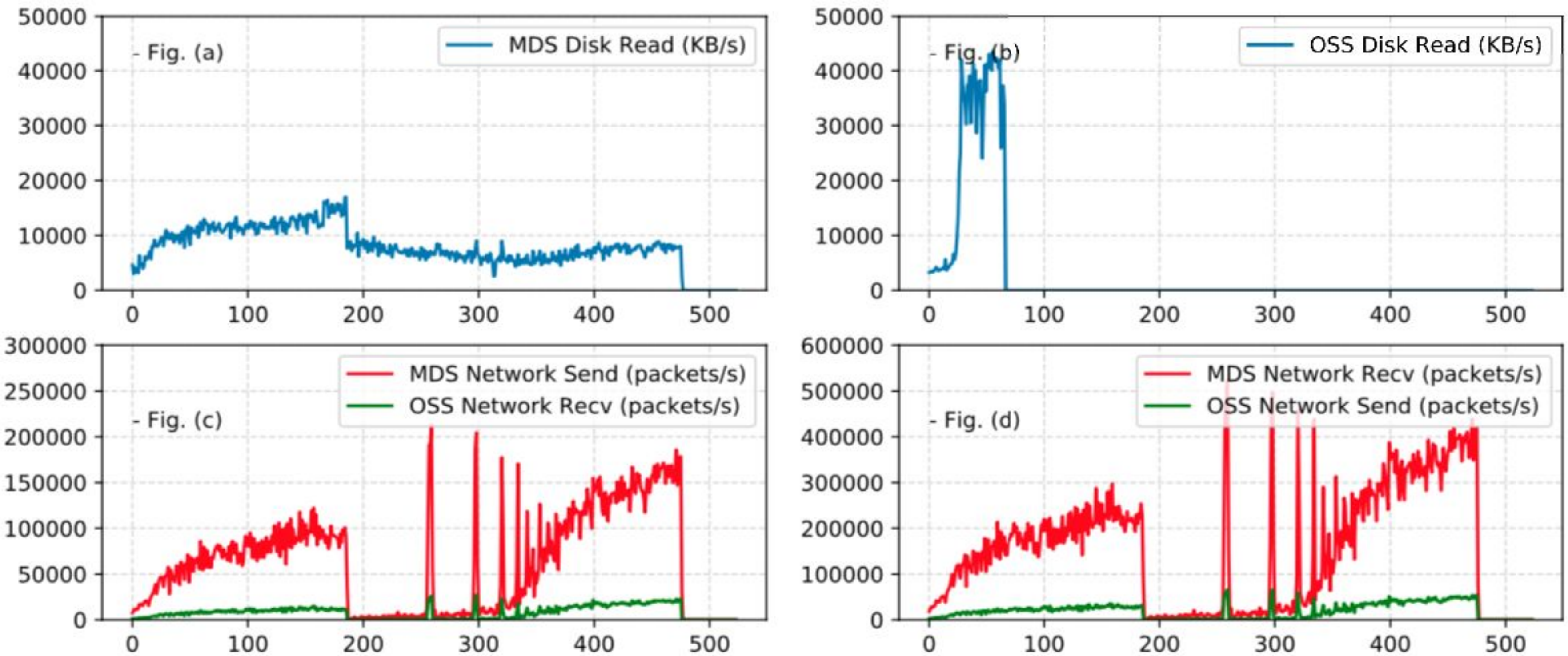
- Scalability
  - How LFSCK reacts to increasing number of OSS nodes (larger Lustre)?
  - The same number of inodes (0.88 Million)
  - LFSCK performance with 2, 4, and 8 OSS nodes
  - With more OSS nodes, LFSCK takes longer time to finish
  - Note that, this results is based on “stripe count = -1”, which stripes files to all the OSSes when possible
    - Production deployment may have better performance
    - For the future, this still can be a problem even for production deployment

TABLE I
EXECUTION TIME OF LFSCK (IN SECONDS) ON LUSTRE WITH DIFFERENT NUMBER OF OSS NODES

# of OSS Nodes	**2-OSS**	**4-OSS**	**8-OSS**
Execution Time	144.8	170.5 (**1.18X**)	218.4 (**1.50X**)

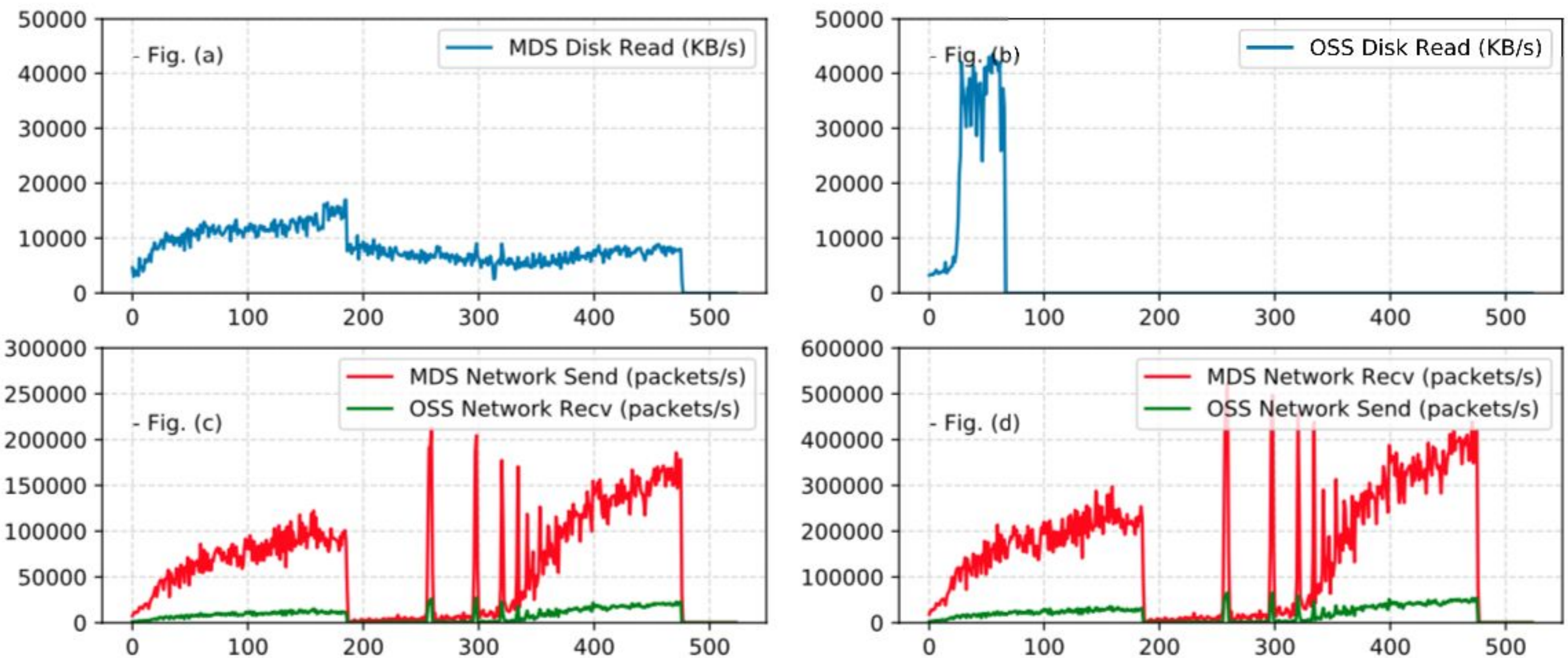
Results and Analysis

- LFSCK performance results
  - Is there potential performance bottleneck in LFSCK implementation?
  - 8 OSS nodes and 1 MDS node
  - Age Lustre with 2.5 million inodes and 4.8 TB of storage
  - Monitor Disk and Network to see what happened during LFSCK



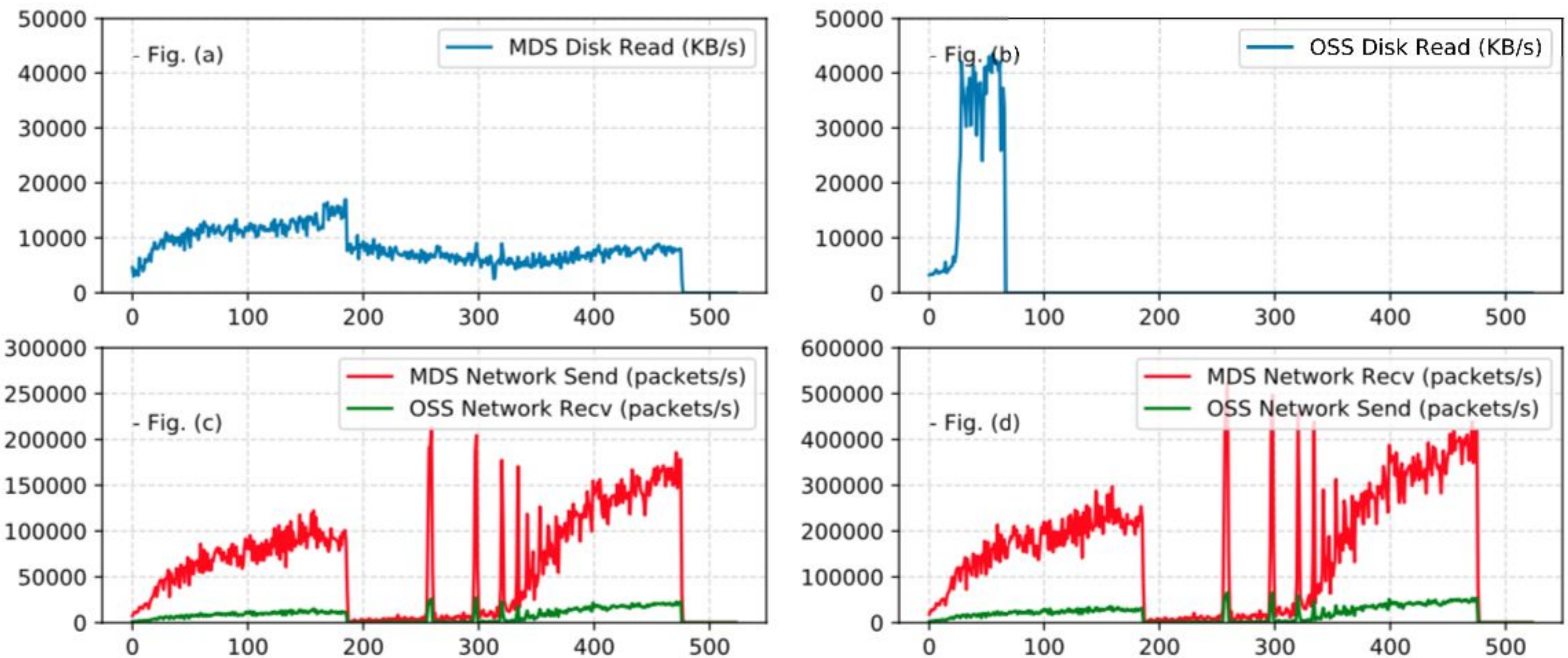
Results and Analysis

- Analysis
  - In MDS and OSS nodes, network and disk are not fully utilized
    - Disk bandwidth is expected to be 100 MB/s
    - Network bandwidth should be more stable and much larger
  - It might because layout checking or namespace checking



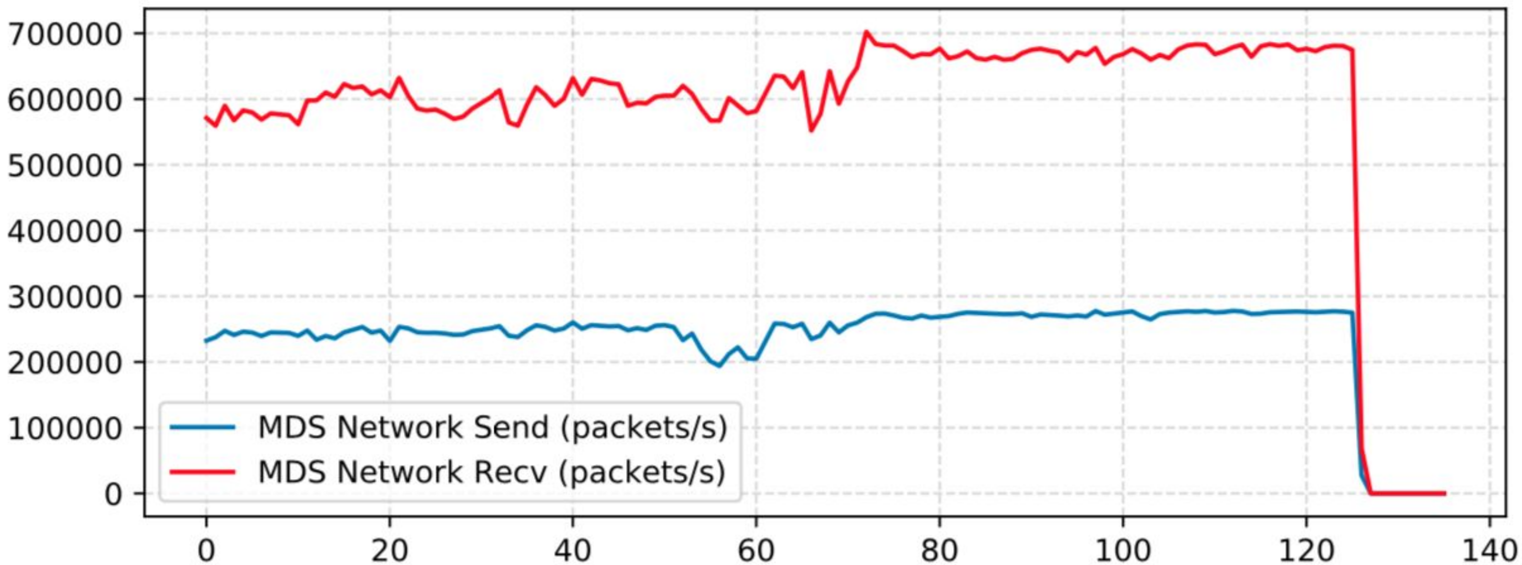
Results and Analysis

- Analysis
  - Look at the network bandwidth closer
    - MDS send around eight times more packages than one OSS receives
    - Plus, MDS receives as many as twice packages as it sends out
  - From the network point of view
    - Such a fan-out ratio indicates MDS can be saturated earlier on the receiving side



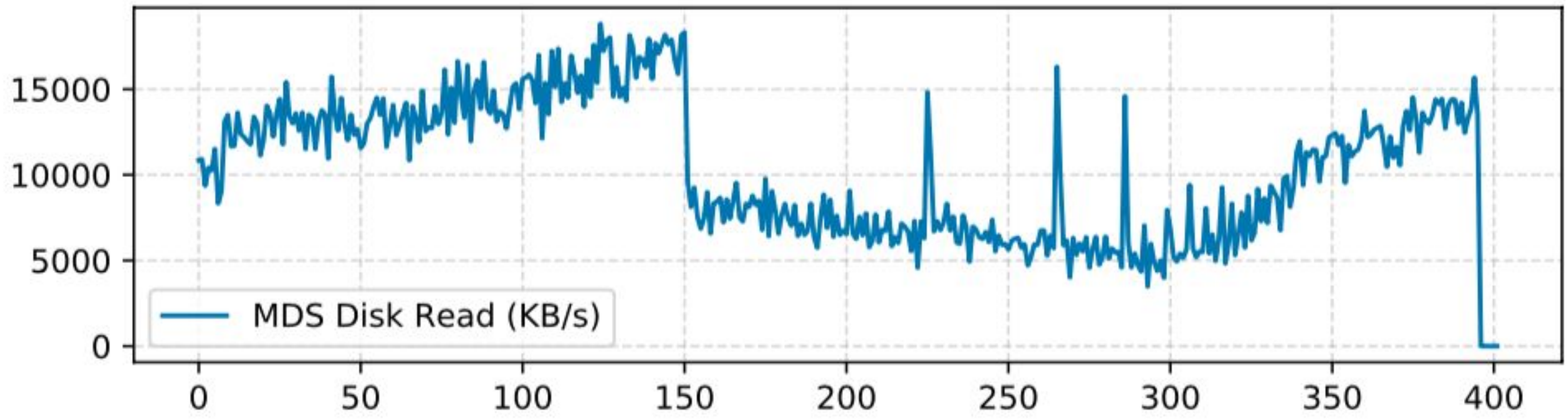
Results and Analysis

- Layout Checking Performance
  - Idea: all MDS and OSS nodes already buffered all metadata in memory
  - Then only network between MDS and OSS affects the performance
  - LFSCK took around 130 seconds to finish (comparing to 500 seconds)
  - Much higher and more stable network band comparing to NOT cached
  - MDS receives double packages compared to what it sends
  - Layout checking is not the bottleneck in previous example



Results and Analysis

- Namespace Checking Performance
  - Namespace checking only happens in MDS node, but it can slow down the layout checking. It seems to be the bottleneck of previous test.
  - Verify this by only running namespace checking
  - Only running namespace checking takes 400 second
  - Incurs the similar low disk bandwidth
  - Confirm that namespace checking is the bottleneck.

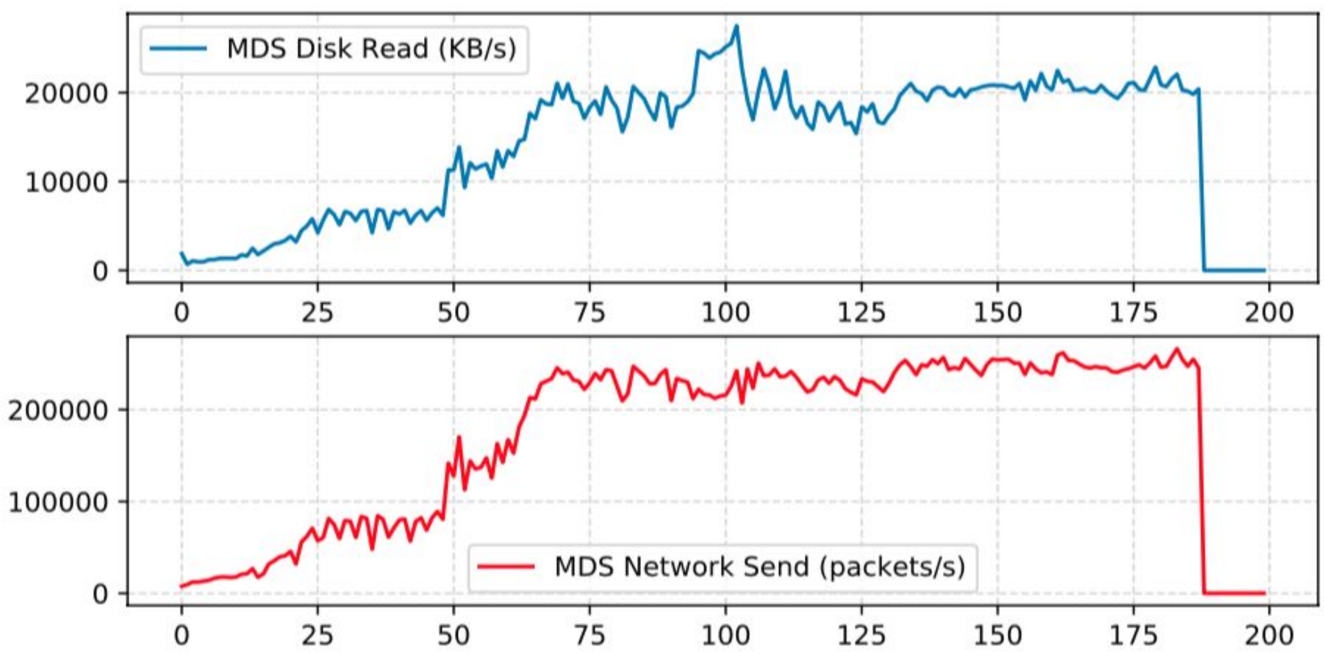


Potentials

- Previous evaluation shows
  - The tight binding between layout and namespace checking in LFSCK design could generate performance bottleneck
  - Namespace checking is too slow due to the slow disk speed.
    - SSD might be helpful
  - Layout checking is faster
    - But, more OSSes can saturate the network earlier and becomes the bottleneck
  - Plus, things are actually dynamic
    - Namespace checking could be temporarily slow because of a huge directory
    - Layout checking could be temporarily slow because of a congested network
- Decouple the tight binding
  - Changing `SCRUB_WINDOW_SIZE` dynamically
  - Changing `LFSCK_ASYNC_WIN_DEFAULT` dynamically

Potentials

- Quick demonstration using current implementation
  - First run layout checking, then run namespace checking
  - Assume layout checking buffers all metadata in memory, indicating an infinite `SCRUB_WINDOW_SIZE` and `LFSCK_ASYNC_WIN_DEFAULT`
  - By simply running them sequentially, LFSCK finishes in 200 seconds
  - Better performance comes from better disk and network bands



Conclusions and Plans

- Conclusions
  - We show that LFSCK still has large room for performance improvement
  - The root cause would be the tight binds of various components
- Current Work and Plans
  - We are running the experiments in more realistic settings
    - Using SSD instead of HDD
    - With more realistic stripe count setting
    - With more realistic aging strategy
  - We are also implementing the proposed optimizations into LFSCK
    - Dynamic parameters setting in runtime
    - Compression of data packages during LFSCK
  - We plan to investigate different strategies to implement LFSCK

Questions?

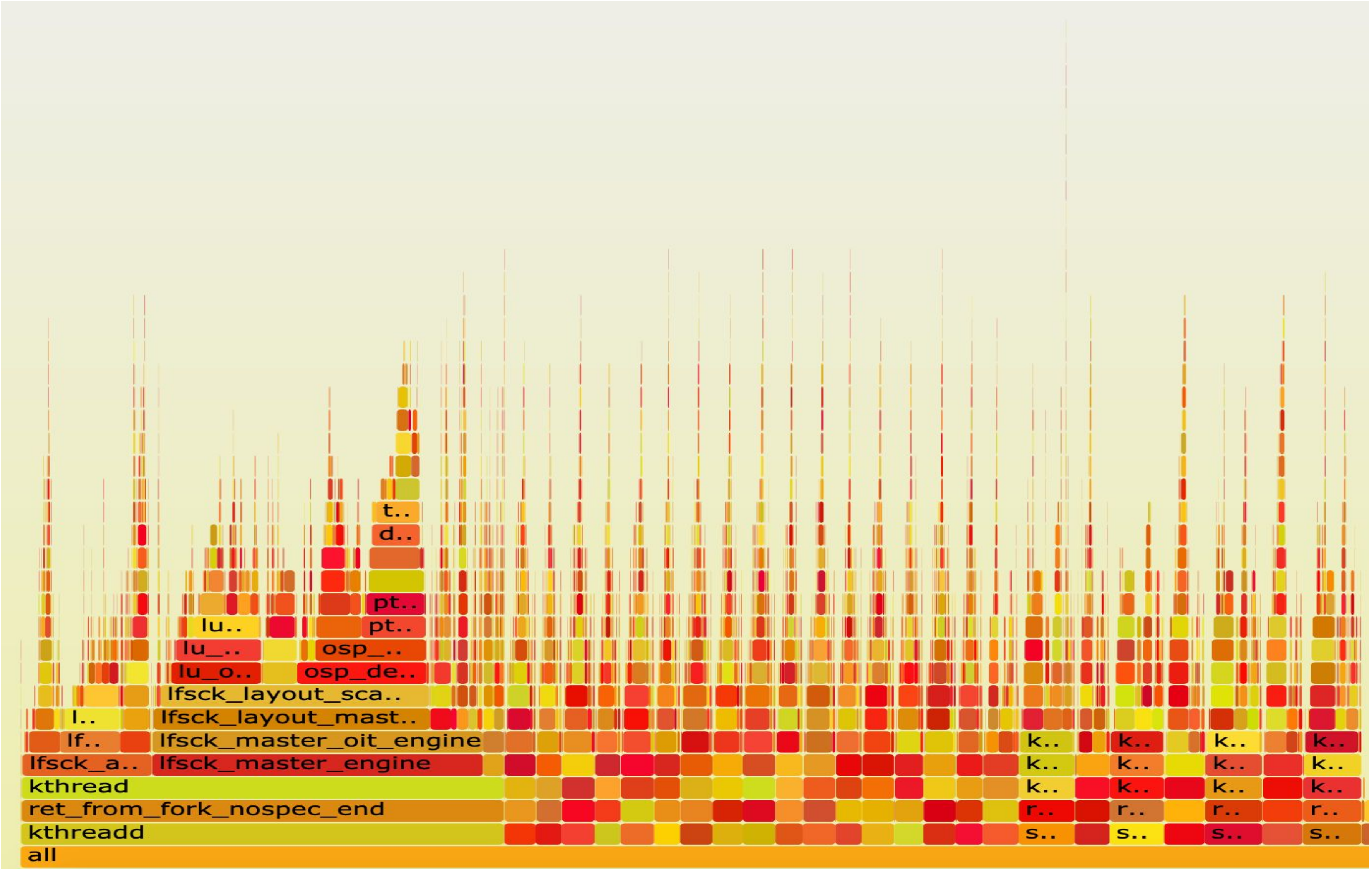
LFSCK Details

- Three essential checks in the first stage of LFSCK
  - It corresponds to the three types of metadata
  - For *mapping metadata*, **[IOScrub]**
    - `FID.local_inode.ext_attr(fid) == FID`
  - For *namespace metadata*, **[Namespace Checking]**
    - `FID.local_inode.ext_attr(linkEA) == parent.FID`
  - For *data layout metadata*, **[Layout Checking]**
    - `FID.Stripe(i).OSS.local_inode.ext_attr(fid) == FID`
- LFSCK runs on both
  - metadata servers (MDSes)
  - object storage servers (OSSes)

LFSCK - Performance Issue

- The performance issue is expected to get worse while HPC is fast moving to “Exascale” and “Data Intensive”
    - More storage servers
    - More complex metadata server design (files on MDT in Lustre 2.11)
    - Much more data files
    - Potentially much larger files
- Then
    - Is it still feasible to run LFSCK in the future Exascale Lustre?
    - Does LFSCK reach the physical limitation of hardware devices?
    - Is it possible to significantly improve its performance?
- We try to answer some of these questions in this study

UNC CHARLOTTE
College of Computing and Informatics


t..
d..
pt..
pt..
lu..
lu_..
osp_..
lu_o..
osp_de..
lfsck_layout_sca..
l..
lfsck_layout_mast..
lf..
lfsck_master_oit_engine
lfsck_a..
lfsck_master_engine
kthread
ret_from_fork_nospec_end
kthreadd
all
k..
r..
s..