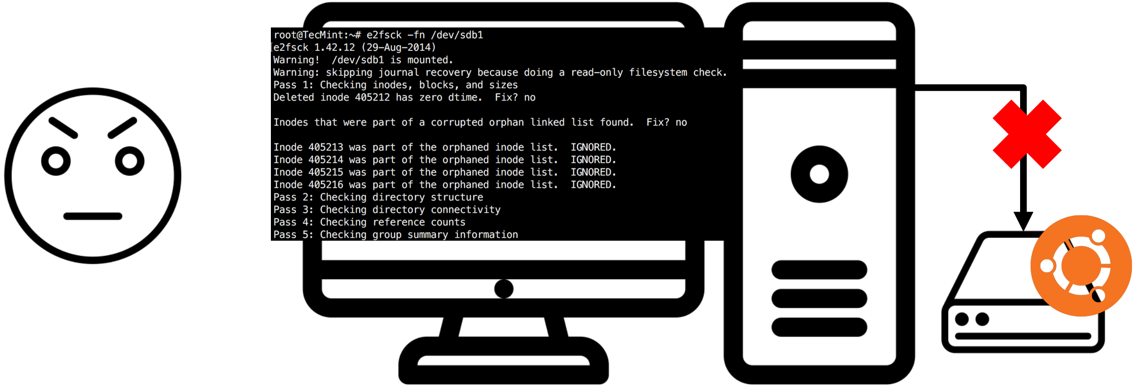
pFSCK: Accelerating Filesystem Checking and Repair for Modern Storage

David Domingo, Sudarsun Kannan
Rutgers University Department of Computer Science

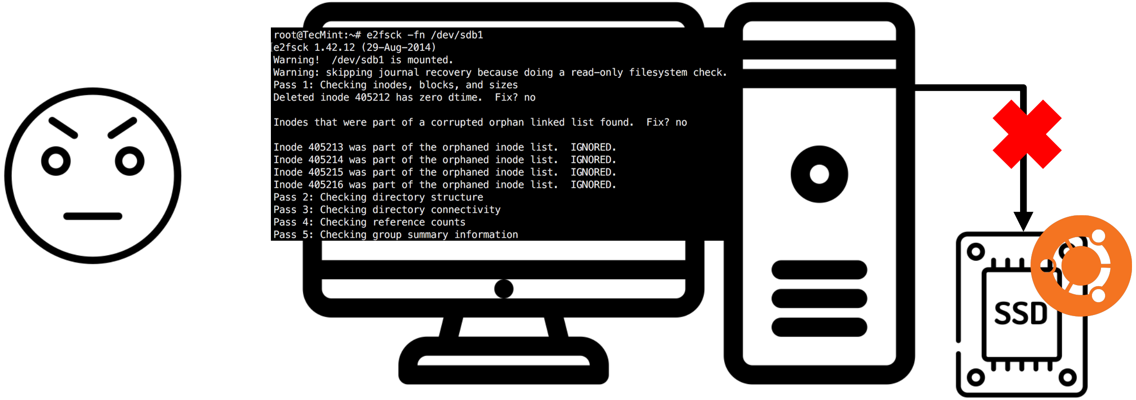
RUTGERS
UNIVERSITY | NEW BRUNSWICK

Storage Issues: A Situation



- You frequently use your personal computer, home server, or production server
- System won't boot after a reboot due to update, restart, or crash
- Forced to run file system checker (ex. e2fsck for EXT4)
- Takes a long time to complete resulting in large downtime and decreased productivity

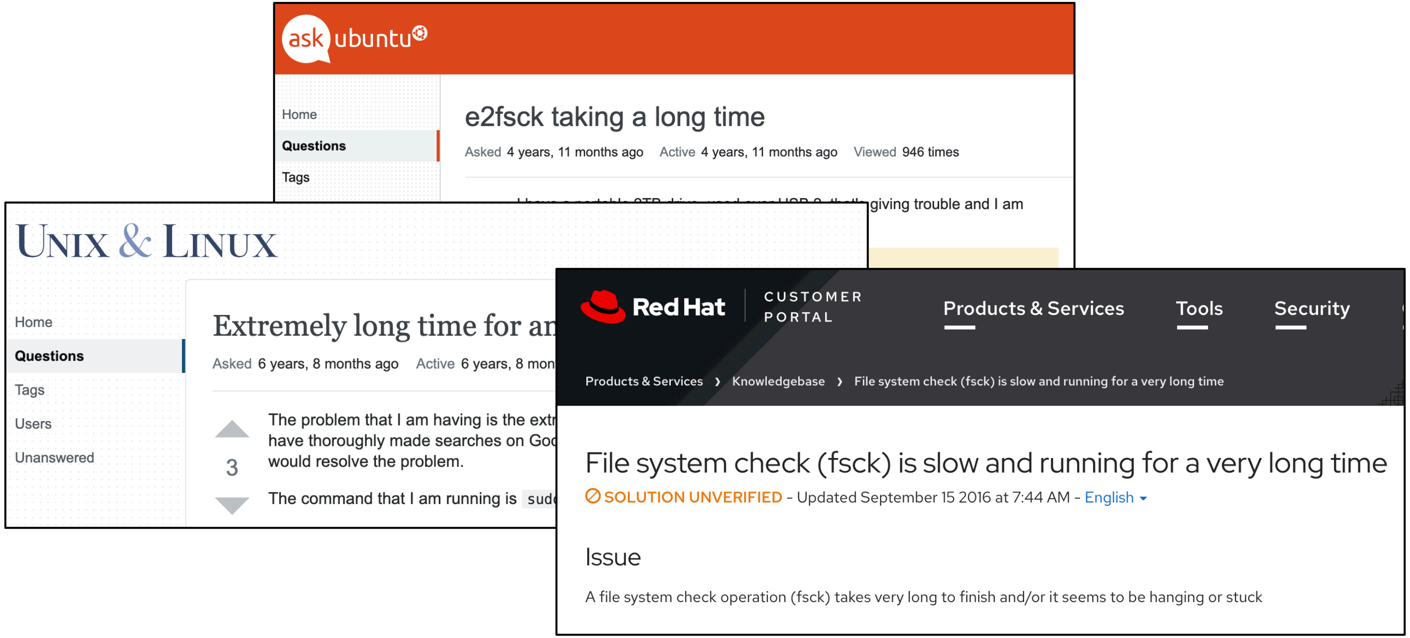
Storage Issues: Faster Storage



- Utilize faster modern storage to reduce runtime and make things easier
- Flash wear increases over time and errors inevitably come up
- Still a hassle and takes long (why?)
- Should run the file system checker proactively to find errors at the cost of availability

Storage Issues: Not a Coincidence

- Frustrating usability is not a rare occurrence
- File system checking has shown to be notoriously slow



Storage Solution: Faster checking with pFSCK

- pFSCK provides faster checking runtimes
  - Parallelizes file system checkers at a fine granularity (e.g. inodes)
  - Ensures correctness through logical reordering
  - Adapts to file system configurations with dynamic thread scheduling
- Shows up to 2.6x improvement over vanilla e2fsck (EXT checker) and 1.8x – 8x improvement over xfs_repair (XFS checker)

Outline

Background: Providing Storage Reliability

- File Systems Checkers
  - Typically used after kernel/security upgrades and crashes
  - Provides ultimate data reliability and recovery

unoptimized to exploit CPU/disk parallelism

- Utilize Modern Storage
  - Larger bandwidth and lower latencies
  - Provides faster scanning and checking

high density storage (e.g. MLC) prone to cell wear and bit corruption

- Utilize Modern Consistency Mechanisms
  - Journaling
  - Copy on Write
  - Erasure Coding
  - Replication

Cannot detect silent bit corruption

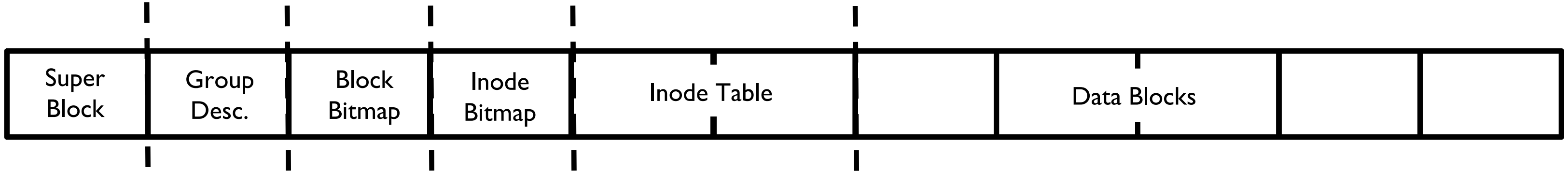
Reconstruction and re-sharding quite time consuming

Not always economically feasible

Traditional file system checking is still relevant!

Background: Disk Layout and Checking

- Linux EXT block group layout



- e2fsck (EXT checker) scans through all file system metadata
- builds own view of file system in order to detect and fix inconsistencies
- Consists of 5 logical passes:
  1. Inodes Pass: Checks inodes (file and directory inodes)
  2. Directories Pass: Checks directories
  3. Connectivity Pass: Checks file reachability
  4. Ref Counts Pass: Verifies link counts for all files
  5. Cylinders Pass: Verifies cylinder group information

Background: Prior Works

- **e2fsck**: Parallelizes file system checking across disks/partitions
- **xfs_repair** : Parallelizes file system checking across allocation groups
- **FFsck (FAST '13)**:
  - Modifies file system and rearranges metadata blocks
  - Provides faster scanning by rearranges metadata blocks
- **ChunkFS (HotDep '06)**:
  - Partitions file system into smaller isolated groups
  - Allows groups to be repaired in isolation
- **SQCK (OSDI '08)**:
  - Uses declarative queries and databases for consistency checks
  - Allows for more expressive fixes with comparable run times

parallelization limited to coarse granularity

requires extensive modification to the file system

requires complete overhaul of file system checker

Outline

Introduction

Background

Motivation

pFSCK Design

Evaluation

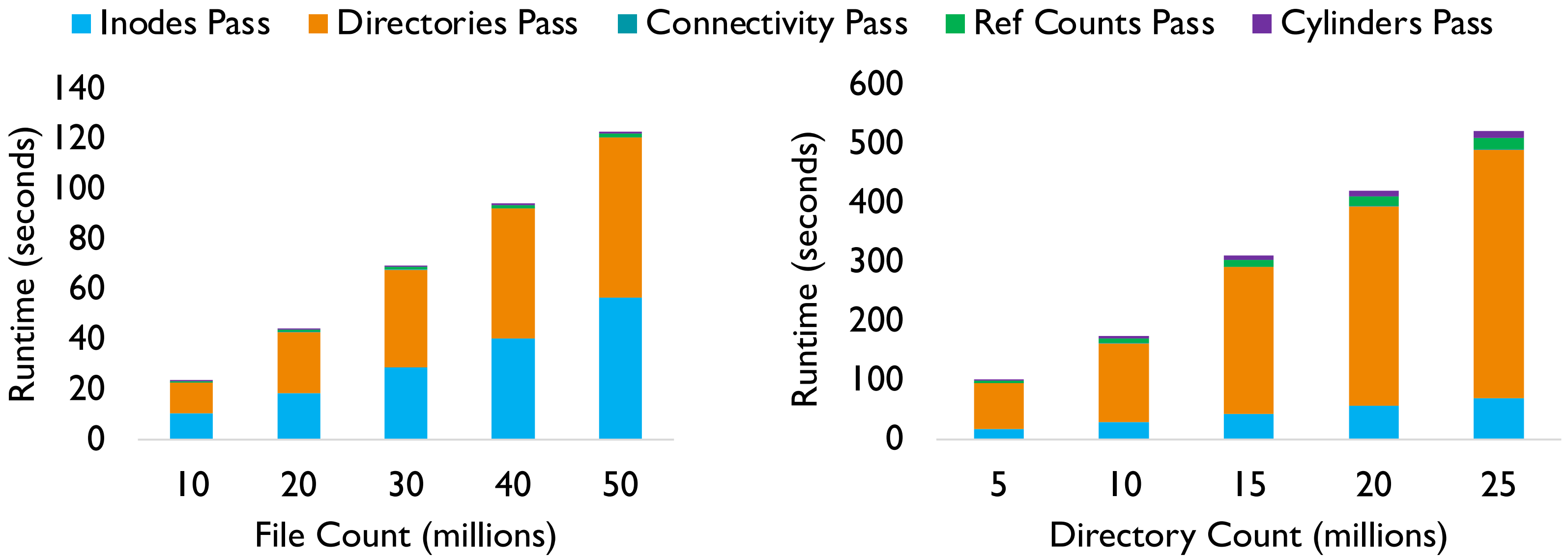
Summary

Evaluating Current e2fsck Performance

- System:
  - Dual Intel® Xeon® Gold 5218 @ 2.30GHz
  - 64 GB of memory
  - 1TB NVMe Flash Storage

- Methodology:
  - e2fsck against 840 GB file systems of varying configurations
    1. Varying file count (file size constant at 12kb, created across 5 directories)
    2. Varying directory count (1 file per directory, each file 24kb)

File System Sensitivity



- Majority of time is spent checking inodes and directory metadata
- Directory-intensive file systems take significantly longer than a file-intensive file system
- Runtime scales linearly with file system utilization

Research Questions

- How to speed up file system checking and repair without compromising correctness?
- How to adapt for different file system configurations?
  - ex. file-intensive vs directory-intensive

pFSCK Key Ideas

- Parallelize file system checking at finer granularity (ex. inodes, directories)

1. Overlap as much independent logical checks within each pass
2. Overlap as much logical checks across passes
3. Reduce contention on shared data structures
4. Efficient management of work for threads across passes

Outline

Introduction

Background

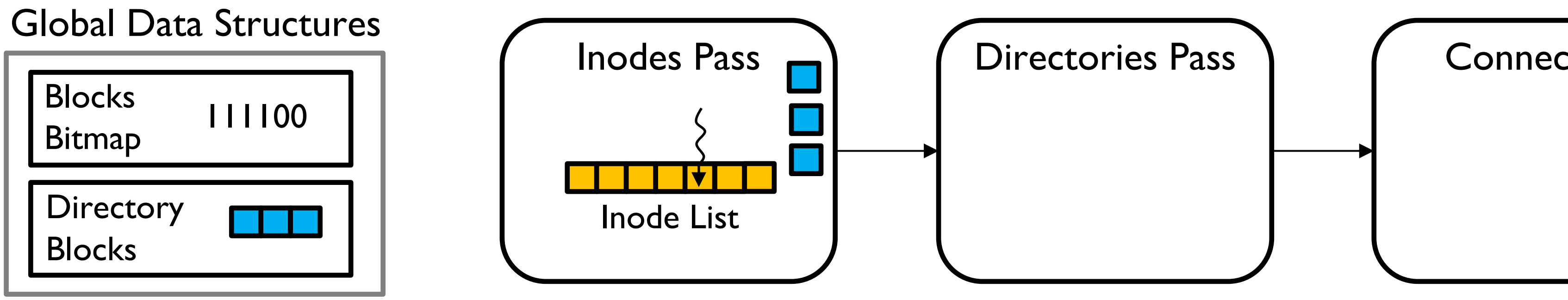
Motivation

pFSCK Design

Evaluation

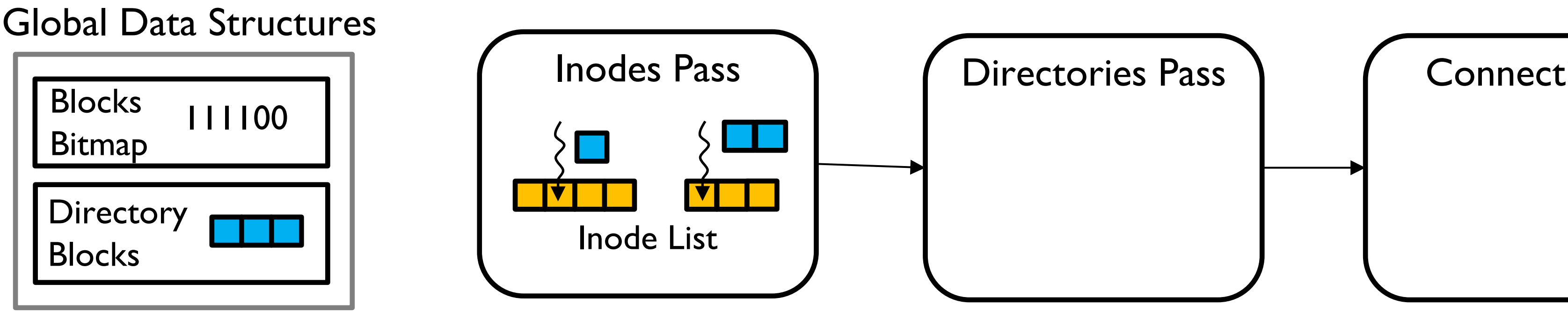
Summary

Serial Execution in e2fsck



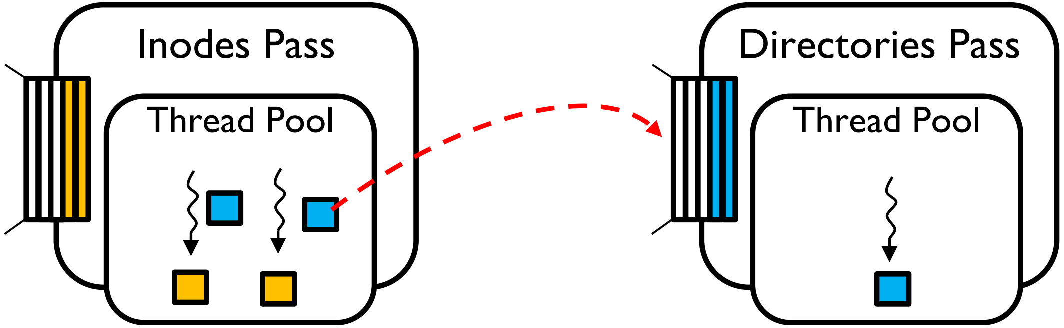
- Serially checks file system metadata (ex. inodes)
- Updates global data structures to generate view of file system
- Generates work for the next pass (ex. list of directory block)

pFSCK Data Parallelism



- Split metadata within each pass into smaller groups
- Uses a pool of threads to check in parallel and generate intermediate lists
- Aggregate lists and repeat
- Critical, unisolated data structures limits potential concurrency (e.g. block bitmap)

pFSCK Pipeline Parallelism

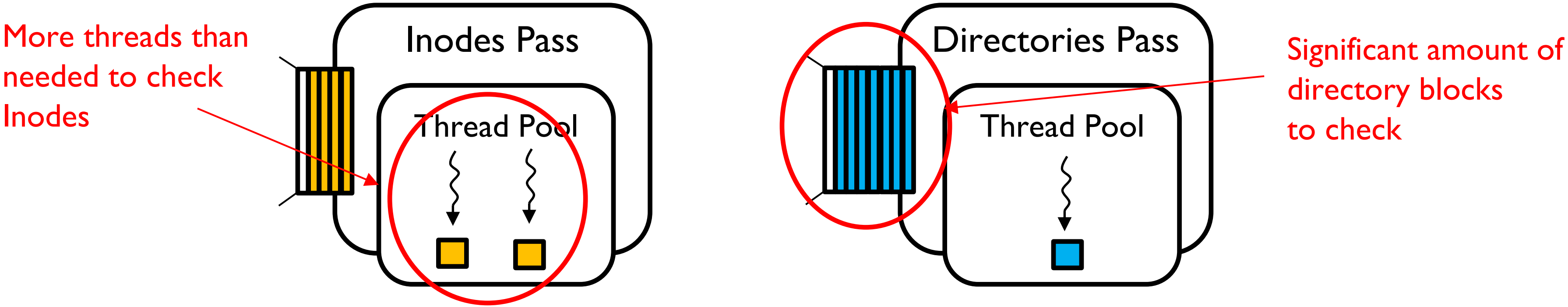


- Allow multiple passes to operate in parallel to hide synchronization bottlenecks
- Turn each pass into independent flows of execution
- Use per pass queues and thread pools
- Continuously feed subsequent passes with metadata
- Do not wait for previous pass to complete (**speculatively carry out future checks**)

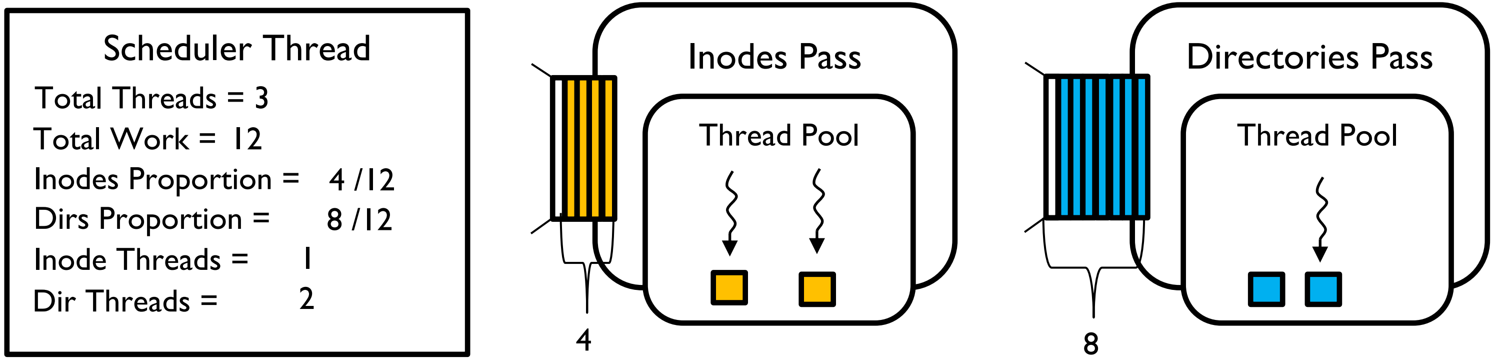
Pipeline Parallelism: Work Imbalance

- Differing metadata densities causes uneven pass queues
- Not straight forward how many threads to assign to each pass
- Example:

  Directory-intensive file system mainly will have more directory blocks to check



Solution: Dynamic Thread Scheduling



- Scheduler thread periodically samples the task queue lengths of each pass
- Calculates relative work among the passes and redistributes threads
- Allows pFSCK to adapt to different file system configurations with differing metadata densities

More in Paper

- Resource-Aware Scheduling for Online Checking Support
- Error handling
- Delayed dependent checks
- Other optimizations

Outline

Introduction

Background

Motivation

pFSCK Design

Evaluation

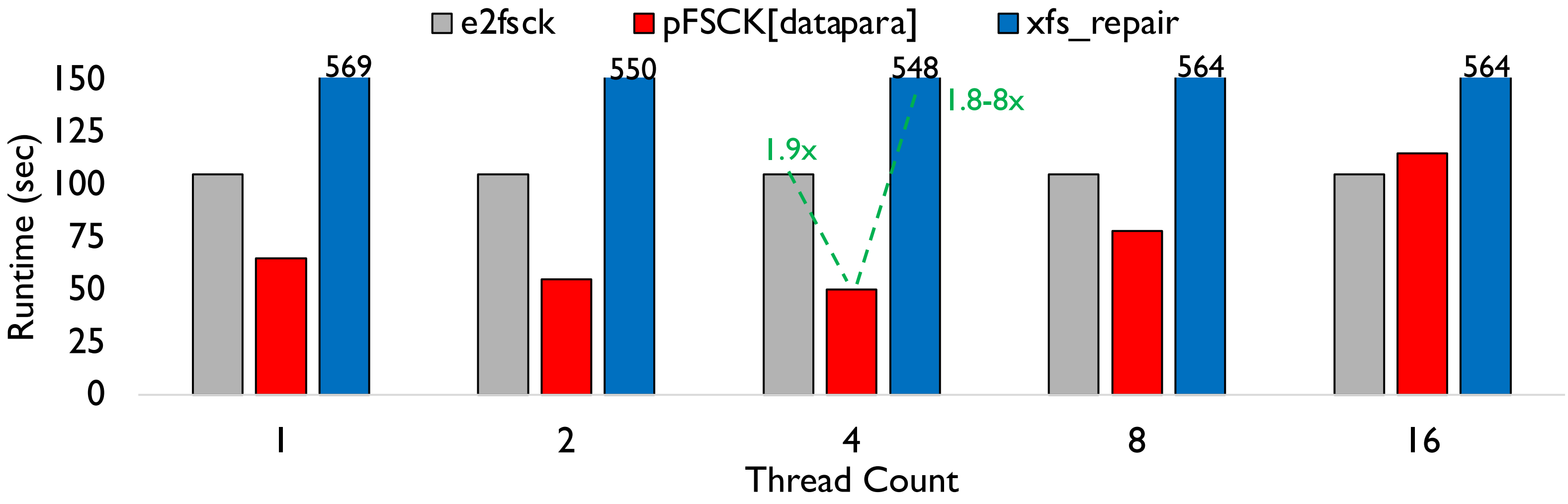
Summary

Methodology

- System:
  - Dual Intel® Xeon® Gold 5218 @ 2.30GHz, 64GB of DDR memory, 1TB NVMe Flash Storage
- Tools:
  - e2fsck (e2fsprogs release v1.44.4)
  - xfs_repair (xfsprogs release 4.9.0)
  - pFSCK (our system)
- File System Configurations:
  - File-Intensive FS (99% files to 1% directories)
  - Directory-Intensive FS (50% files to 50% directories) (see paper)

Evaluation: Data Parallelism

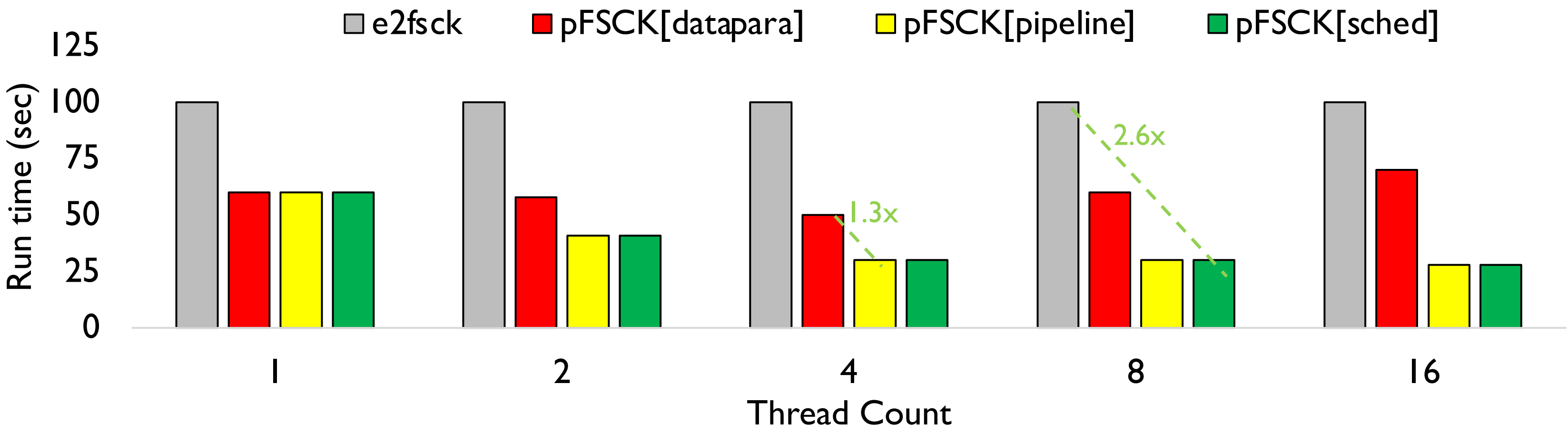
- pFSCK's data parallelism compared to vanilla e2fsck and xfs_repair



- Improves performance over vanilla e2fsck by up to 1.9x (4 Threads)
- Improves performance over xfs_repair by up to 1.8x – 8x with same thread count
- Contention on shared structures limits data parallelism scaling

Evaluation: Pipeline Parallelism and Dynamic Thread Scheduler

- Pipeline parallelism and dynamic thread scheduling compared to just data parallelism



- Pipeline parallelism increases performance by 1.3x over just data parallelism
- Dynamic Thread Scheduler automatically identifies optimal thread assignment
- pFSCK[sched] automatically improves performance by up to 2.6x over vanilla e2fsck

More in Paper

- More File System Configuration Analysis
- Memory Usage and I/O Throughput Analysis
- Resource-Aware Scheduling Performance
- Performance on SSD
- Error Fixing Performance

Outline

Summary

- pFSCK provides fine grained parallelism for file system checking
- Data parallelism allows more metadata to be checked at a time
- Pipeline parallelism enables parallelism across passes
- Dynamic thread scheduler adapts to file system configuration
- pFSCK is provides up to 2.6x performance over vanilla e2fsck

Thank You!

David Domingo
djd240@cs.rutgers.edu

Sudarsun Kannan
sudarsun.kannan@cs.rutgers.edu