# Log Structured File System

Surya Suresh

# Background

- 1990s seeing technology advancements
  - Processors
    - Increases in processor speed
  - Memory
    - Increase in memory size
  - Disk
    - Improvements in cost and capacity
    - Limited performance in transfer bandwidth and access time
- All 3 components are important in file system design

# LFS Design Motivation

- Technology and types of workload
  - How can current technological advances improve file system design?
  - Small file access workloads?
  - Large file access workloads?
- Problems with disk performance
  - Can the number of seeks be reduced?
- Problems with existing file systems
  - Data spread out, causing many seeks
  - Synchronous writes

# Goals of LFS

- Reduce the number of reads by caching files in main memory
- Cache small write modifications to later persist as one large block to disk
- Persist changes in an append-only fashion
- Have cleaning policies to compact data and free up space

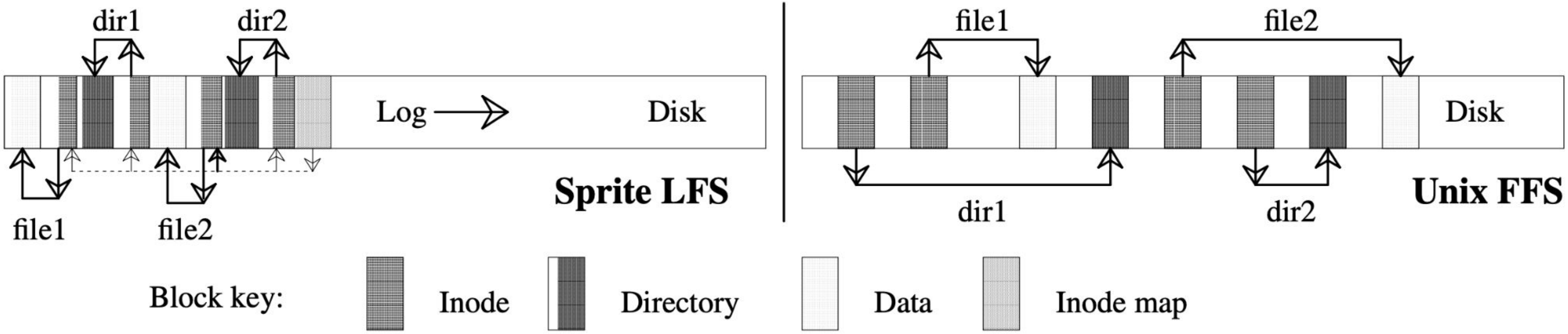
# Design Implementation: Indexing Structures

- No bitmap or free list needed

| Data structure | Purpose | Location | Section |
|---|---|---|---|
| Inode | Locates blocks of file, holds protection bits, modify time, etc. | Log | 3.1 |
| Inode map | Locates position of inode in log, holds time of last access plus version number. | Log | 3.1 |
| Indirect block | Locates blocks of large files. | Log | 3.1 |
| Segment summary | Identifies contents of segment (file number and offset for each block). | Log | 3.2 |
| Segment usage table | Counts live bytes still left in segments, stores last write time for data in segments. | Log | 3.6 |
| Superblock | Holds static configuration information such as number of segments and segment size. | Fixed | None |
| Checkpoint region | Locates blocks of inode map and segment usage table, identifies last checkpoint in log. | Fixed | 4.1 |
| Directory change log | Records directory operations to maintain consistency of reference counts in inodes. | Log | 4.2 |

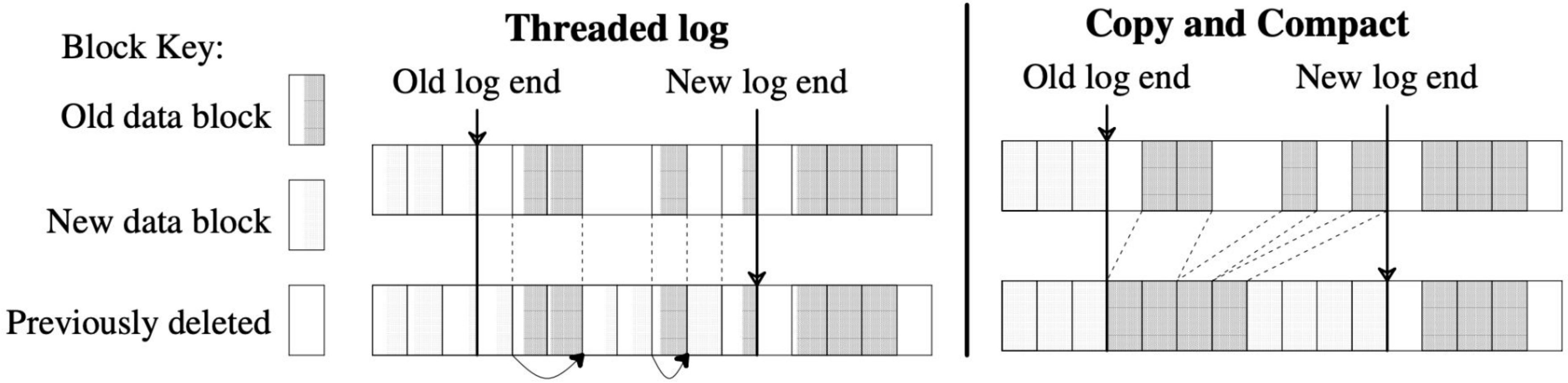
# Design Implementation: Log Structure

- Treat disk as a circular buffer, always appending changes
- Broken up into chunks called segments



# Segment Cleaning

- Combination of threading and copy and compact techniques were used

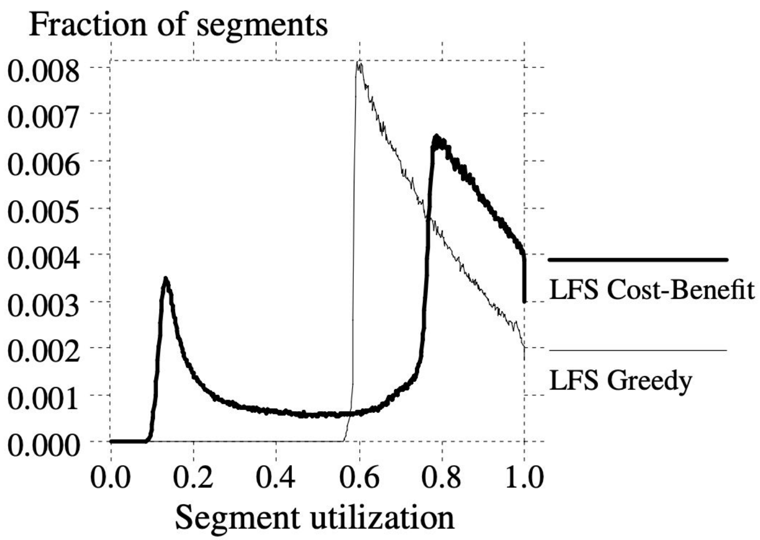


# Crash Recovery

- Checkpoints
  - Position in log where all file system structures are consistent and complete
  - Checkpoint region updates to contain addresses of these structures
  - Upon reboot, structures will loaded into memory using checkpoint
- Roll Forwards
  - Scan through log segments written after latest checkpoint
  - Helps recover as much data as possible

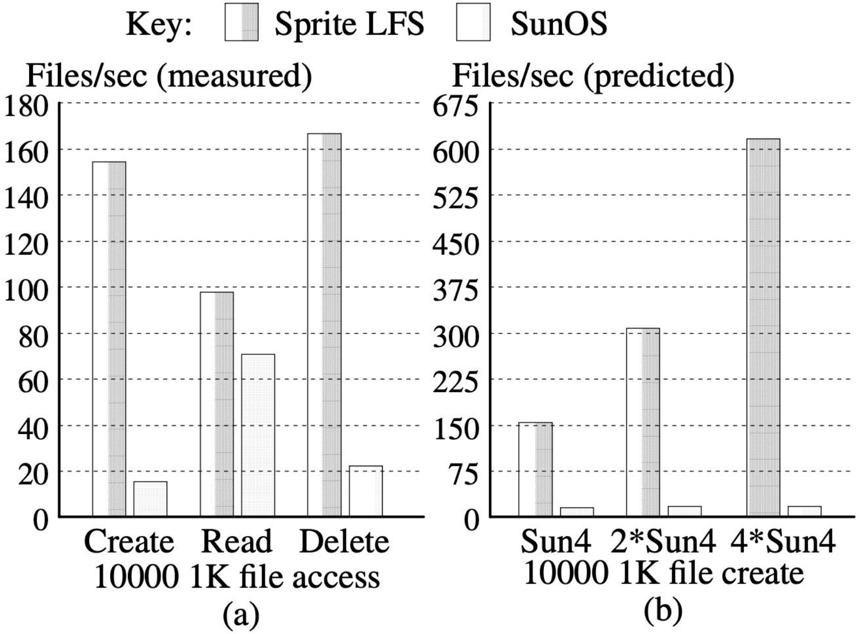
# Performance Eval: Segment Cleaning Policy

- Cost-benefit policy calculates benefit (amount of free space reclaimed and time space will be free) and cost (read segment and write live data)
- Helps clean cold segments at higher utilization
- Better to clean cold segments than hot segments



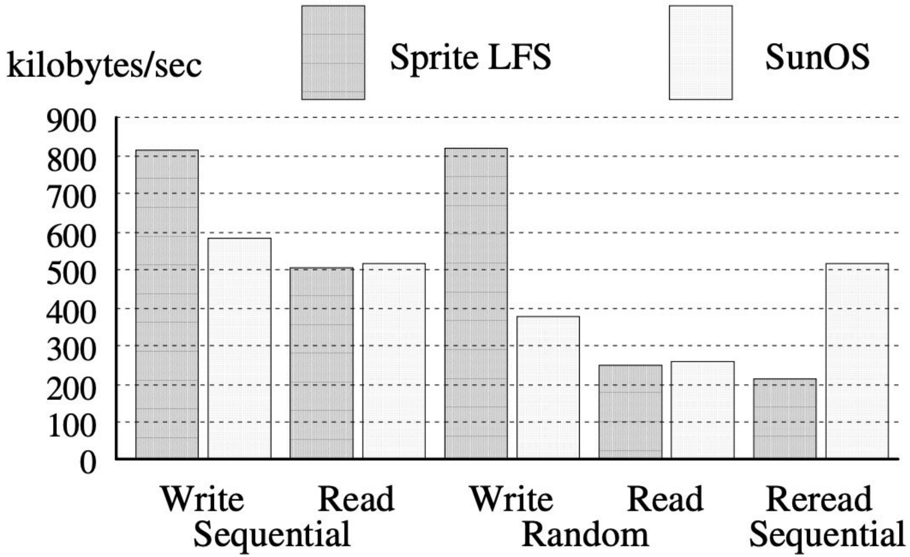
# Performance Eval: Small File Microbenchmark

- Performance for lfs is much higher than ffs for creates and deletes, little better for reads



# Performance Eval: Large File Microbenchmark

- Large file optimizations weren't in mind
- Seem to perform better apart from rereads



# Discussion

- Is the log structured file system a viable design used today?
- What if ffs cached files like lfs did? Do you think it would be better? Worse? Same?
- Is lfs too reliant on the assumption that main memory will soak up reads?