Log-structured File Systems

By

R V Pavan Kumar

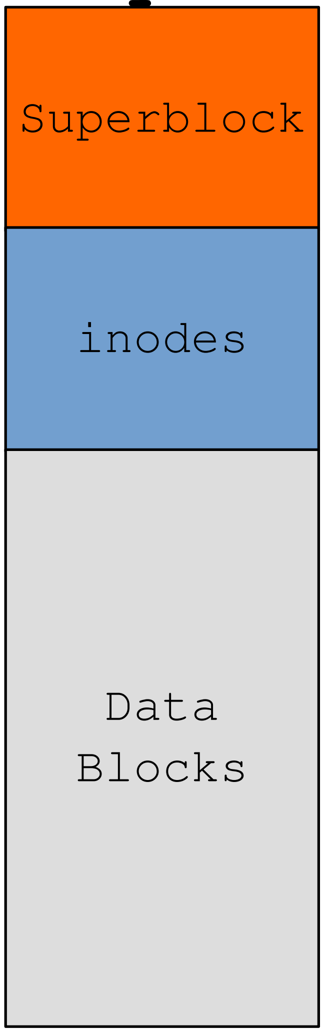
Outline

- Unix File System
- Berkley Fast FS
- Improvements
- Evaluation
- Log-Structured FS
- Contributions
- Evaluations

Unix File System

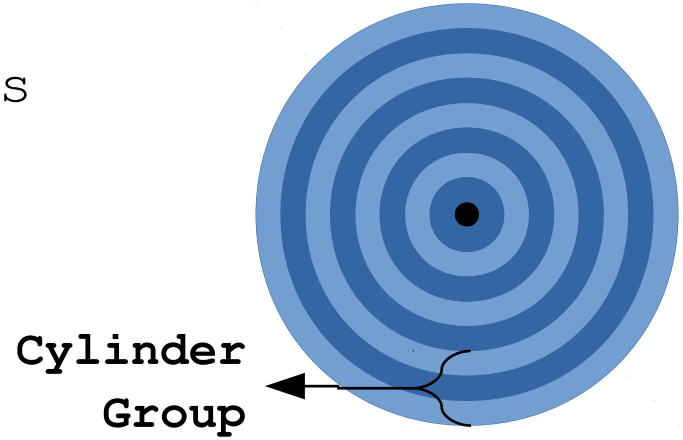
- Bell Labs by Ken Thompson
- Block size – 512Bytes
- Linked list of free blocks
- Problem: Poor Performance
  - Low throughput – 2-4% of max bandwidth
  - Randomization of data blocks
  - Unorganized free list

Layout



Berkley Fast File System

- UCB CSRG, 1984
- Filesystem organization
  - Cylinder group – Set of consecutive cylinders
  - Superblock – replicated for recovery
  - Bitmap – free list
  - Inodes – static allocation
  - Block size – 4098 or 8196 Bytes
- Considers physical disk geometry



Improvements

- Optimal Utilization
  - Fragments – 2,4 or 8 per block
  - Last block can be fragmented
- Free space reserve(10%) – maintain throughput
- Data Locality
  - Global layout policies
    - Cluster related data
    - Spreading out unrelated data
  - Local allocation routine
    - Always find near-by free blocks
    - Make room for locality

Evaluation

- 20-40% improvement in bandwidth
- 10x faster reads
- Good Performance on large sequential writes
- Writes are 50% slower than reads
  - Overhead of allocating blocks
- Stable performance

The Design and Implementation of Log-Structured File System
ACM SOSP 1992

John K Ousterhout

Mendel Rosenblum

University of California
Berkeley

Motivation

- CPU speeds **Vs**

  Disk access speeds **Vs**

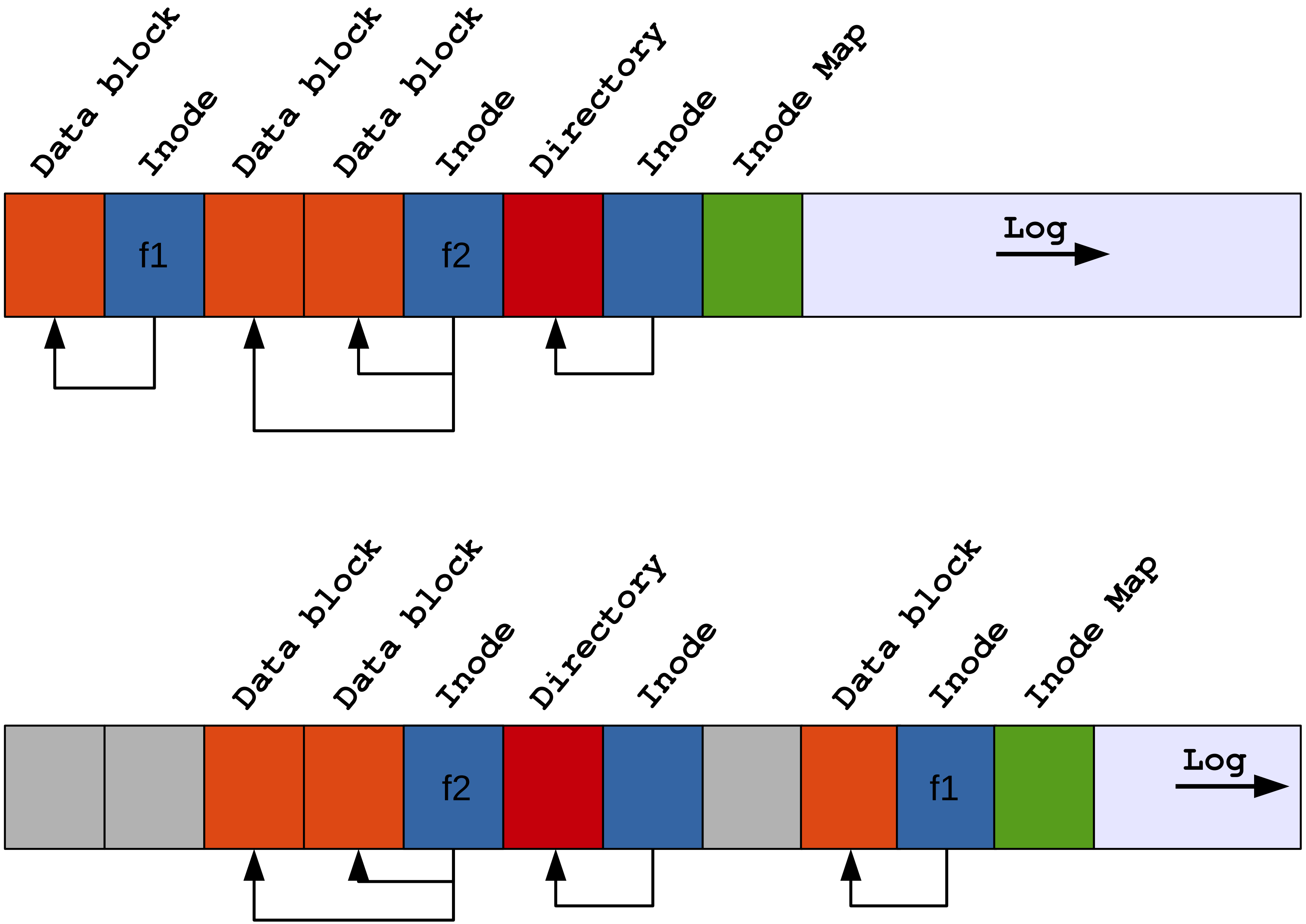
  Main Memory
- Greater Main Memory – Absorbs most reads
- Disk traffic dominated by writes
- Problems with existing FS
  - Poor efficiency of small-file access
  - Spread of information
  - Synchronous metadata writes

Overview

- Basic Idea: Asynchronous writes of large sequential data
- Write data includes all file system information
- Maximum utilization of disk bandwidth
- Key challenges
  - Retrieve information
  - Maintaining free space to write

File Access

- File has a Inode
- Inode
  - File attributes
  - 10 Data Blocks
  - Indirect blocks
  - No fixed location
- Inode Map – written to the log as blocks
- Fixed checkpoint region will have all inode map locations

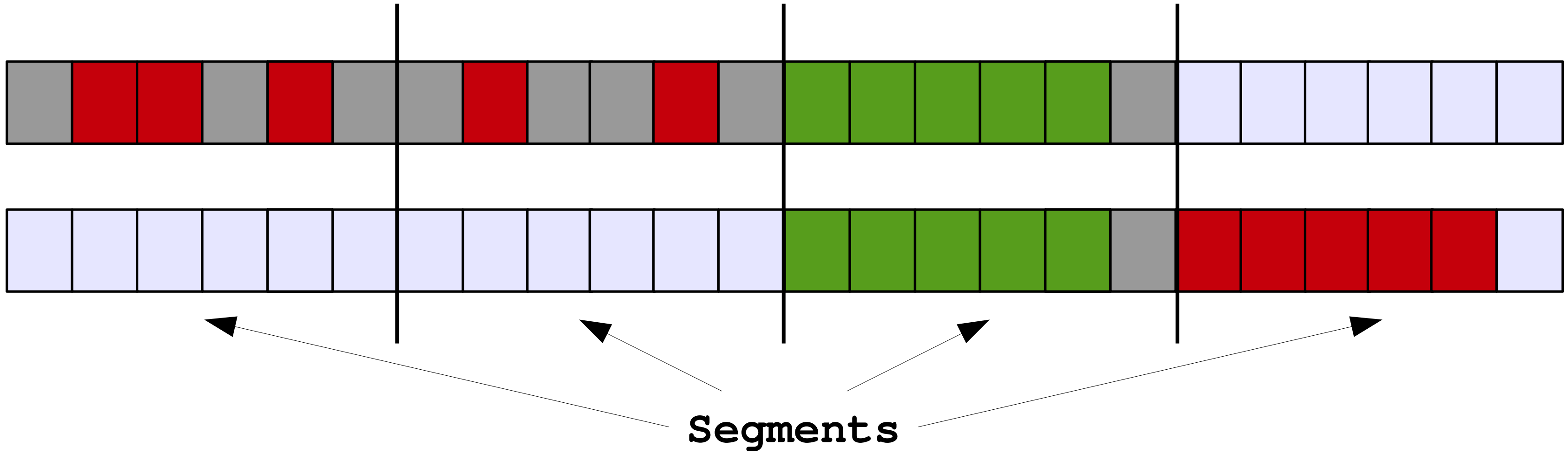

Data block
Inode
Data block
Data block
Inode
Directory
Inode
Inode Map
f1
f2
Log
Data block
Data block
Inode
Directory
Inode
Data block
Inode
Inode Map
f2
f1
Log

Free space management

- Fragmented free space due to overwrite and deletion
- Threading new data through free blocks
  - Further increases fragmentation
- Copying live data into compacted form
  - Complete lock of file system
  - Unnecessary movement of long lived files

Segments

- Combination of threading and copying
- Data written to a segment as log
- Segment cleaning: Copy live data out of segment
- Skip segments filled with long lived live data.
- Segment size – 512KB or 1MB



Segment Cleaning

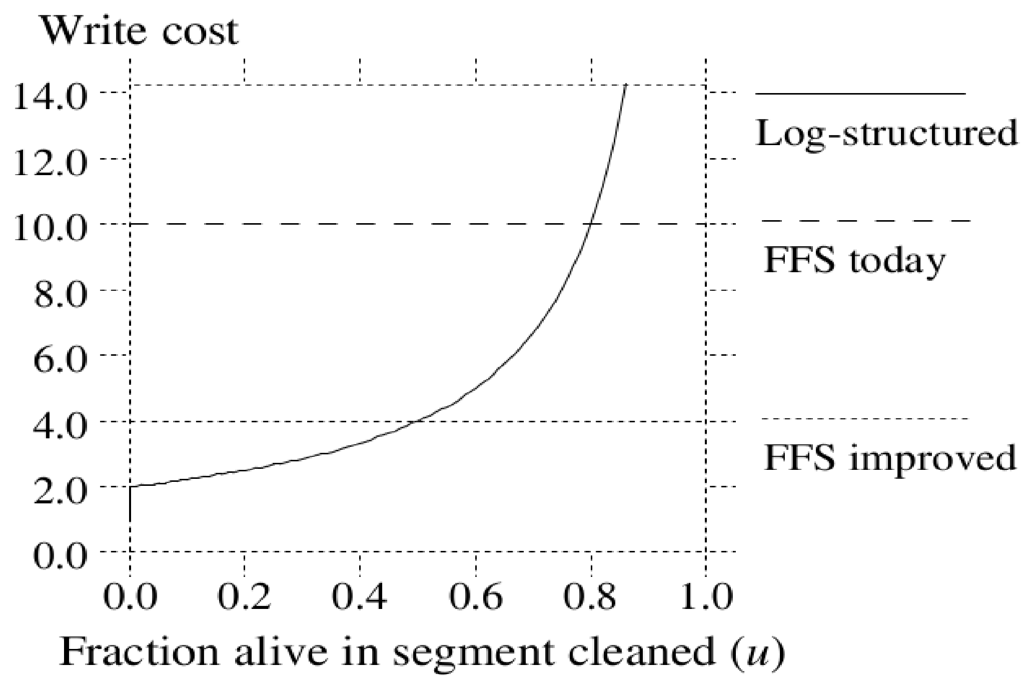
- Identify live blocks
- Update File inode
- Segment summary block solves these problems
  - Per block file uid and block number
  - Partial writes will result in multiple summary blocks
  - Little overhead during writing
  - Useful for crash recovery and cleaning
- File UID = version number + inode number

Segment cleaning policies

- When should it execute ?
  - Number of segments < threshold value1
- How many segments it should clean ?
  - Number of segments > threshold value2
- Which segments it should clean ?
  - More fragmented – not best
- How should live blocks be grouped ?
  - Locality based
  - Age based

Policy Selection

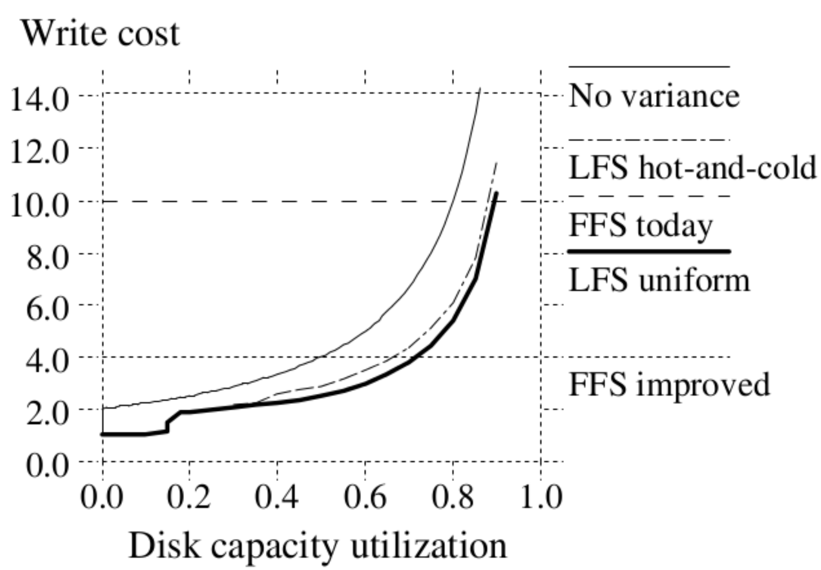
- Write cost: average time disk is busy per byte of new data written
- Write cost = $\frac{total\,bytes\,read \wedge written}{new\,data\,written} = \frac{2}{1-u}$
- U: utilization of the segment $0 \leqslant u < 1$

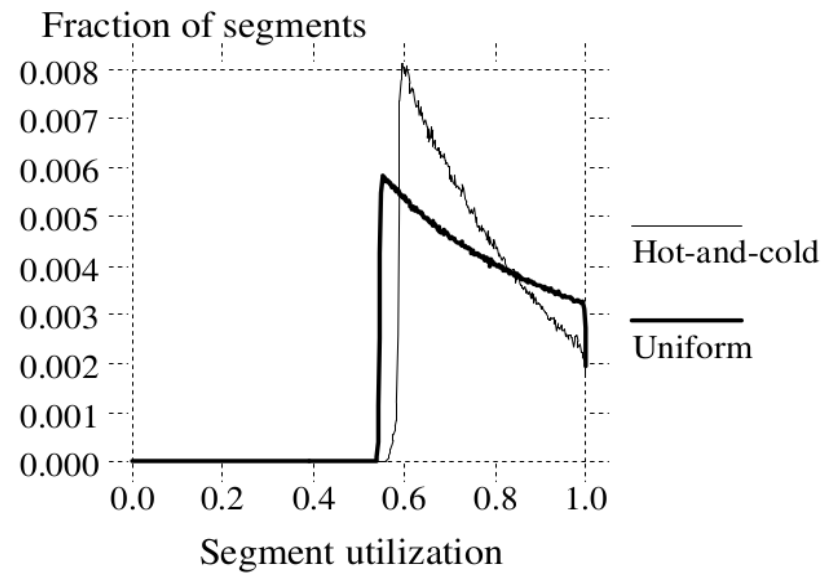


Simulation

- Fixed number of 4KB files
- Overwrite a pseudo-random file selected by
  - Uniform
    - all file are equal
    - No re-ordering while writing
  - Hot-and-cold
    - Hot group: 10% files, 90% selected
    - Cold group: 90% files, 10% selected
    - Order based on age
- Greedy policy – least utilized segments to clean

Simulation Results

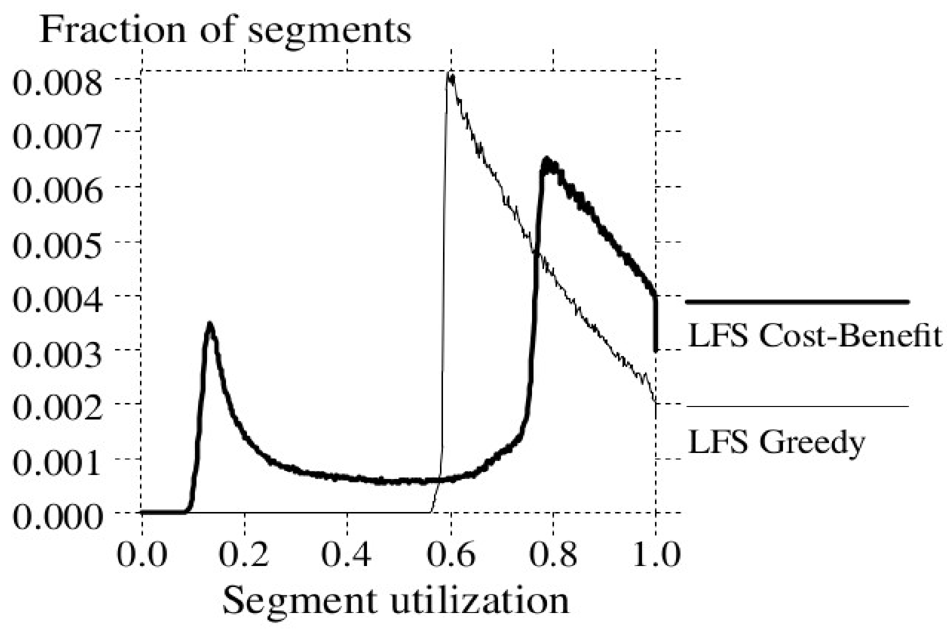




- LSF Uniform – lower write cost
- LSF Hot-and-Cold was performing worse even under the consideration of locality
- Space in Hot and cold segments must be valued differently

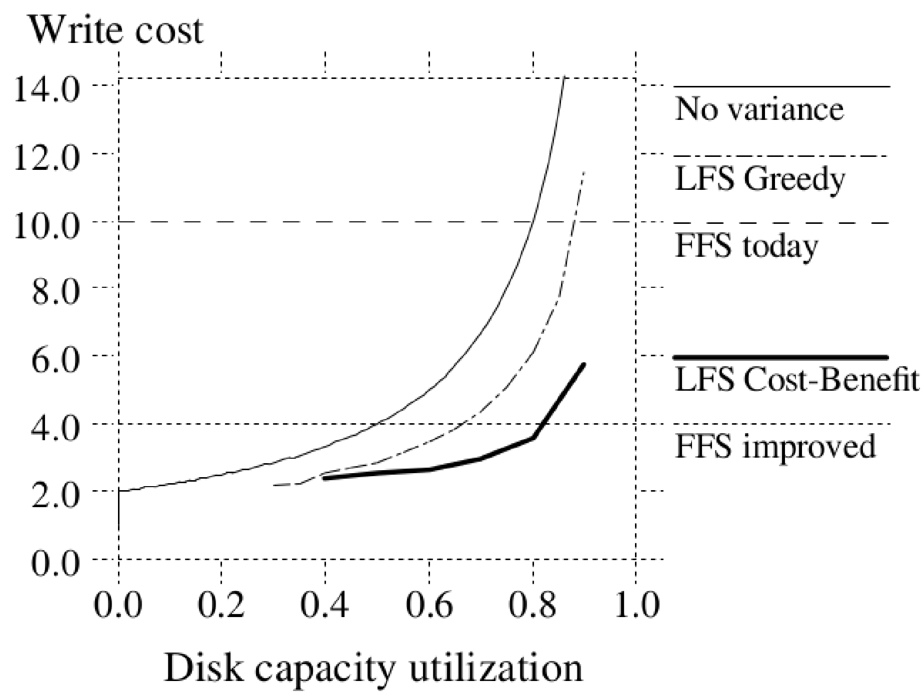
LFS Cost-Benefit

- The value of a segment is based on its stability
- Assumption: Older data will remain unchanged
- High $\frac{benifit}{cost} = \frac{free\ space\ generated * age\ of\ data}{cost} = \frac{(1-u)*age}{1+u}$
- Segment age = latest modified time among all segment blocks



Results

- Bimodal distribution of segments
- Cold segments at u=75%
- Hot segments at u=15%
- 50% reduction in write cost compared to greedy
- Segment usage table
  - No of live bytes per segment
  - Most recent modified time

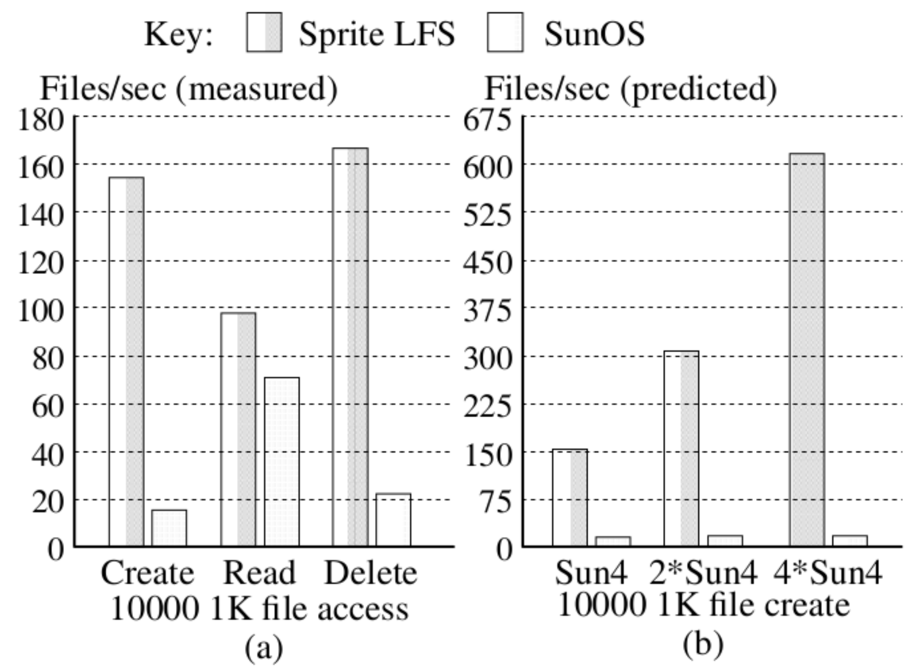


Crash Recovery

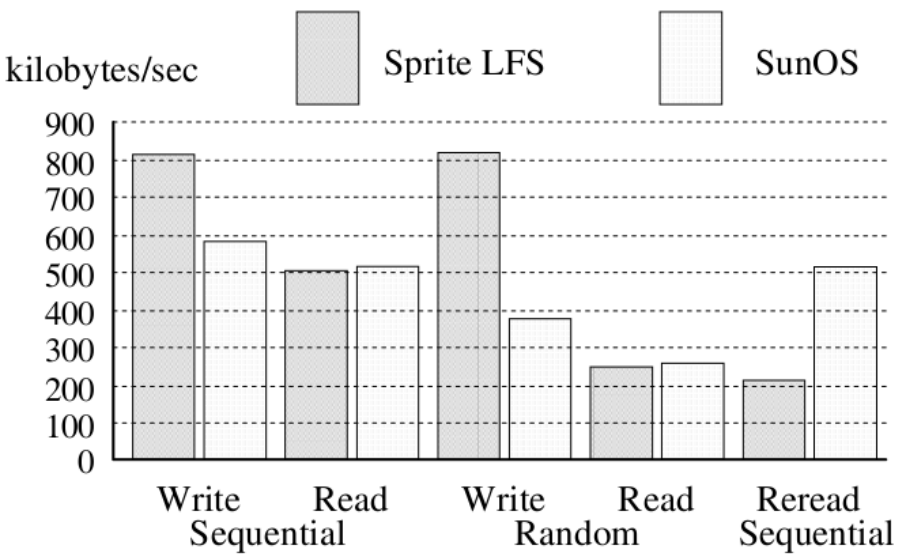
- Check points
  - Write out all modified information
  - Update fixed checkpoint region
    - Current time
    - Pointer to last segment written
  - Two checkpoint regions
  - Periodic intervals (30s), unmount and shutdown
- Roll-forward
  - Scan the log from last written segment
  - Update directories, inodes, inode map and segment usage blocks
  - Updates the checkpoint region
- Directory operation logs – consistency between directory entries and inodes
  - Operation code(create, link, rename, unlink)
  - I-number of directory and position within it
  - Name and I-number
  - New reference count

Micro-benchmarks

- Sun 4/260
  - 16.67MHz
  - 32MB
  - 300MB, 1.3MB/s, 17.5ms
- Compared LFS against Unix FFS
- Block size - FFS 8KB, UFS 4KB/1MB(segment)
- 10000, 1KB files
  - create -> read -> delete
- LFS kept disk 17% busy
- FFS kept disk 85% busy



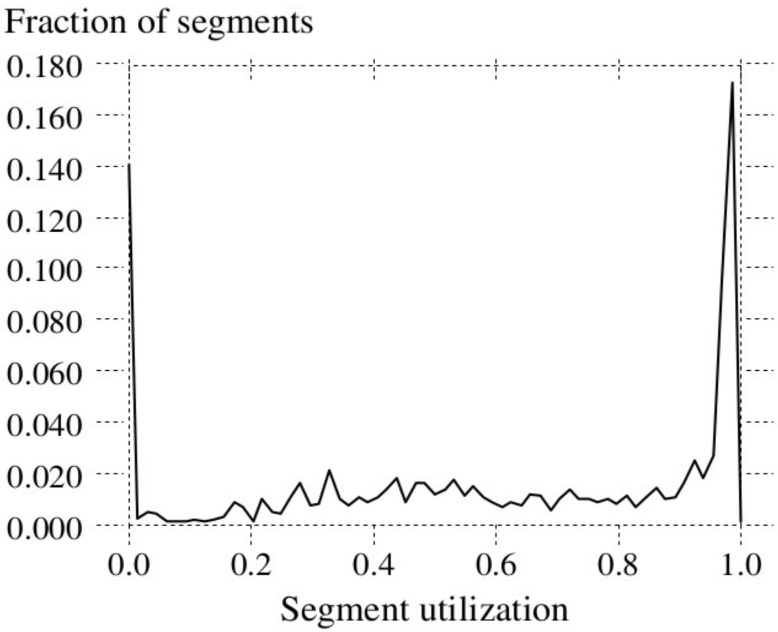
Large-file performance



- Benchmark
  - Write sequential, 100MB
  - Read sequential
  - Write randomly, 100MB
  - Read randomly, 100MB
  - Read sequential
- LFS has best write performance in all cases
- LFS sequential reading after writing randomly is poor
- LFS – temporal locality
- FFS – logical locality

Cleaning Overhead

Write cost in Sprite LFS file systems								
File system	Disk Size	Avg File Size	Avg Write Traffic	In Use	Segments Cleaned	Segments Empty	u Avg	Write Cost
/user6	1280 MB	23.5 KB	3.2 MB/hour	75%	10732	69%	.133	1.4
/pcs	990 MB	10.5 KB	2.1 MB/hour	63%	22689	52%	.137	1.6
/src/kernel	1280 MB	37.5 KB	4.2 MB/hour	72%	16975	83%	.122	1.2
/tmp	264 MB	28.9 KB	1.7 MB/hour	11%	2871	78%	.130	1.3
/swap2	309 MB	68.1 KB	13.3 MB/hour	65%	4701	66%	.535	1.6



Later work

“An implementation of Log-structured filesystem for Unix”- Margo Seltzer,Keith Bostic,Marshal Kirk McKusick,Carl Staelin - 1993 USENIX

File systems for flash memories and SSD's where wear-leveling is required.
Examples: YAFFS(Yet Another Flash File System) and JFFS(Journaling Flash File System)

Thank You

Q?