Florida International University

The Design And Implementation of a Log-Structured File System

Luis Useche
lusec001@cs.fiu.edu

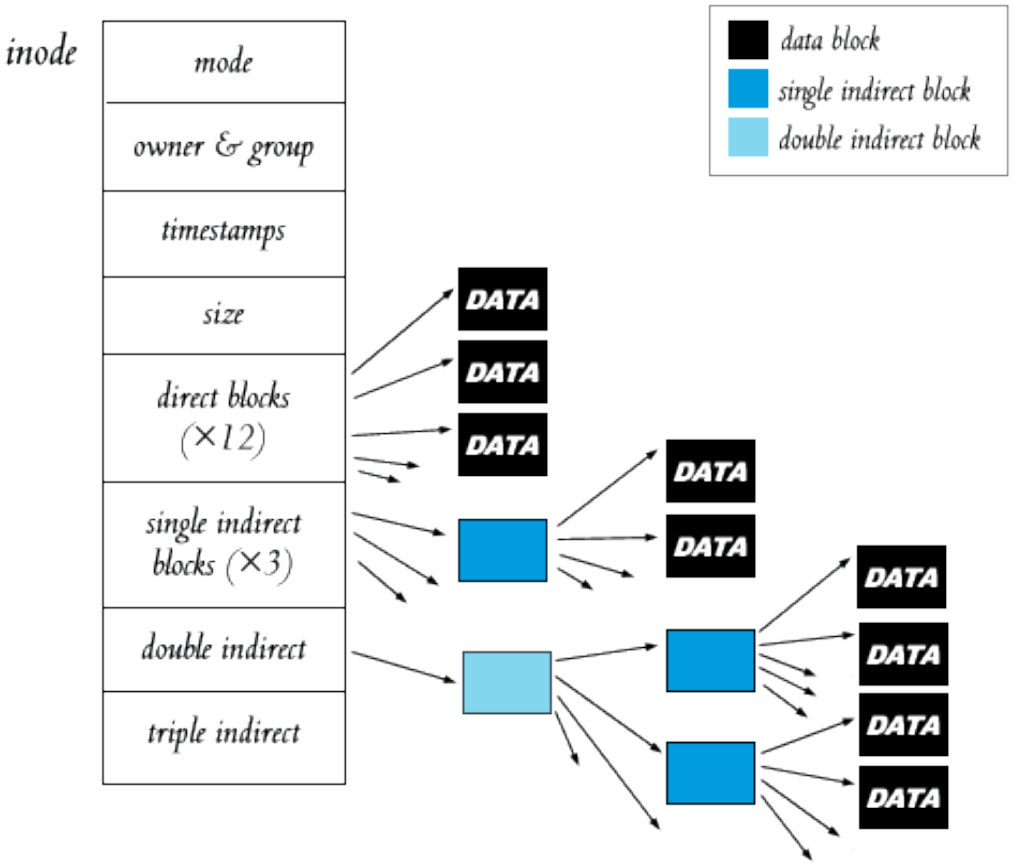
11/2/06

Overview

- Introduction
- Motivation
- Assumptions
- Log-Structured File System
- Experience with the Sprite LFS
- Conclusions
- References

Introduction

- Unix File Systems (UFS): How does it work?



Motivation

- Problem: Data stored in the Unix-like file systems is fragmented on disk.
- Solution: In Log-Structured file systems the new data is appended to end of the log avoiding write seeks.

Assumptions

- Large main memory is available.
  - Large read cache satisfy more effectively the read requests
  - Disks are dominated by writes
- Office and engineering applications tend to accesses small files

Log-Structured File System(LFS)

- Changes are cached, and then wrote in a large write operation.
- Append new information at the end of the log
- The log is the only structure on disk

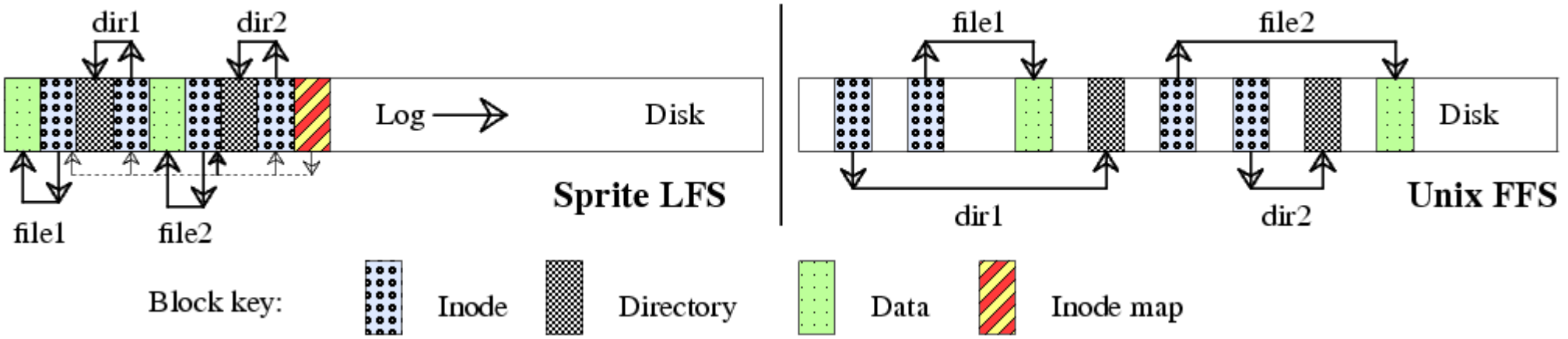
Problems Introduced

- Are sequential scans required to retrieve information?
- Log Structure need extents free space in disk

Data Location and Reading

- Goal: Match or exceed the read performance of UFS.
- The basic structures to index the files are identical to those used in UFS
  - The inodes are not in a fixed location: how to find it?: INODE MAP

LFS vs UFS Write Example



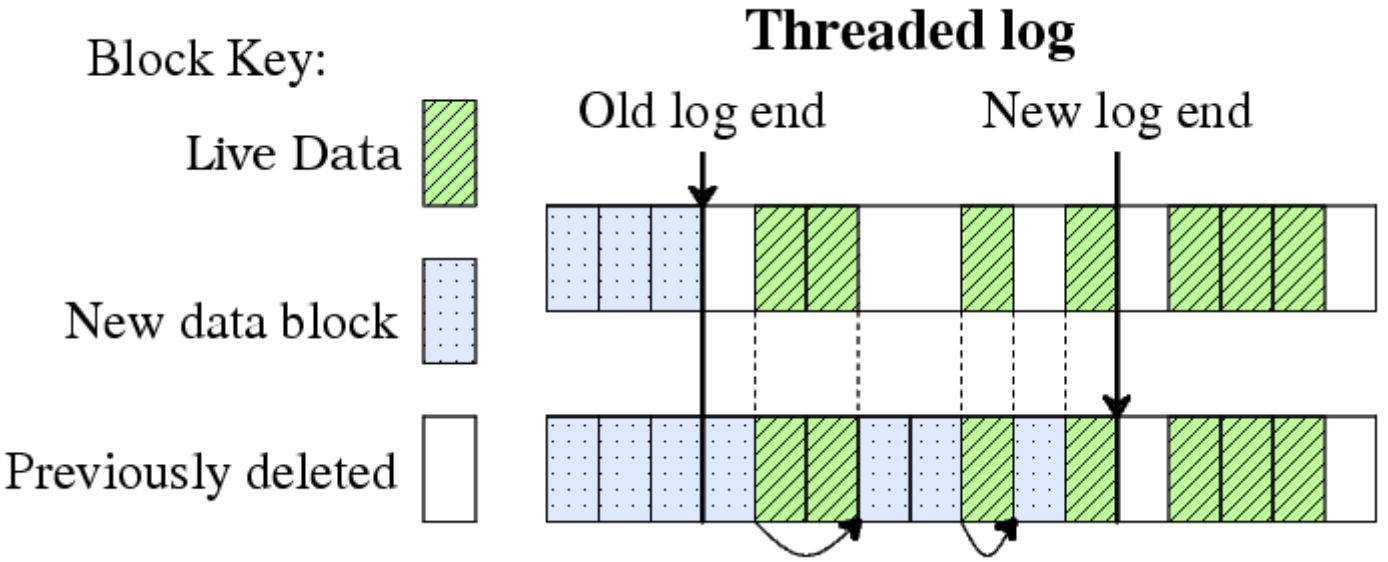
Writing 2 files: dir1/file1 and dir2/file2

Free space management

- Goal: Maintain large free extents for writing new data
- Two possible choices:
  - **Threading**
  - **Copying**

Threading

- Leave the live data in place and thread the log through the free extents

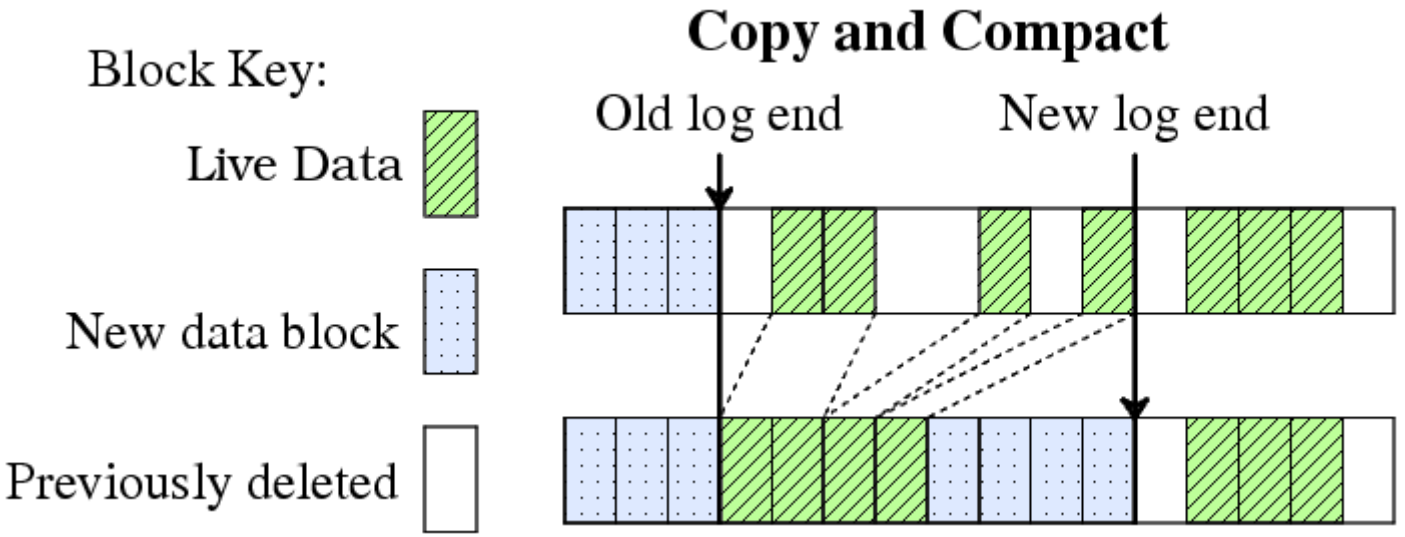


Threading: Advantages and Disadvantages

- Advantages: Simplicity, low overhead.
- Disadvantages: Free space severely fragmented thus large contiguous writes won't be possible.

Copying

- Compact the live data in order to free enough space to the new data

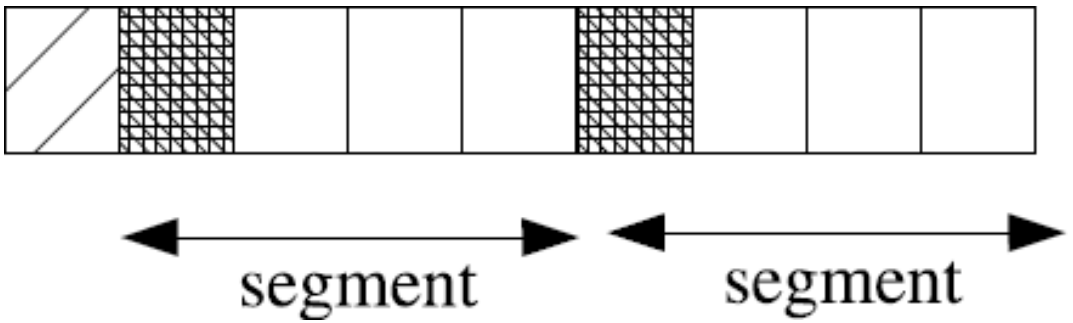


Copying: Advantages and Disadvantages

- Advantages: Avoid fragmentation, the data can always be written in large contiguous pieces.
- Disadvantages: Copying overhead reduce the bandwidth used to write new data.

Free space management Solution

- Introduce a new structure call Segment.



Segment Restrictions

- All live data should be copied out of the segment before reusing it, this process is called Segment Cleaning.
  - The modifications of a particular block is appended at the end of the log and the inode structure is modified.

Segment Restrictions (cont)

- The log is threaded on an segment-by-segment basis.
- The segment size is large enough that the transfer time is much greater than the cost of seek to the beginning of the segment.

Segment Cleaning mechanism

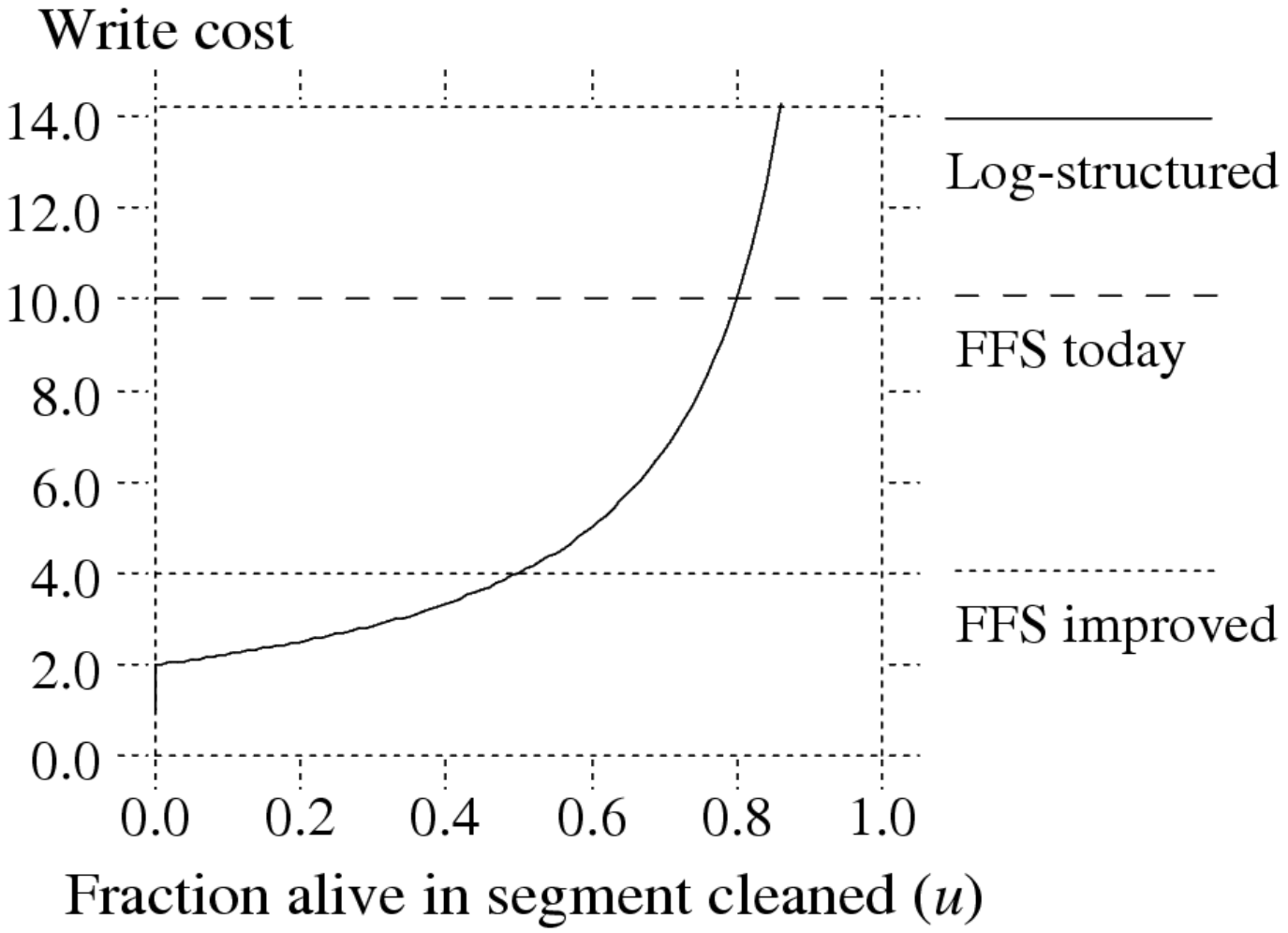
- Goal: release space to write new data.
- There are some cleaning policies:
  - When should the segment cleaner execute?
  - How many segments should it clean at a time?
  - Which segments should be cleaned?
  - How should the live blocks be grouped when they are written out?

Policies Evaluation

- The policies are evaluated with the write cost value.
  - Utilization: Amount of live data in a particular segment.

$$
\begin{aligned}
write\ cost &= \frac{total\ bytes\ read\ and\ written}{new\ data\ written} \\
&= \frac{read\ segs + write\ live + write\ new}{new\ data\ written} \\
&= \frac{N + N \cdot u + N \cdot (1-u)}{N \cdot (1-u)} \\
&= \frac{2}{1-u}
\end{aligned}
$$

Write cost function



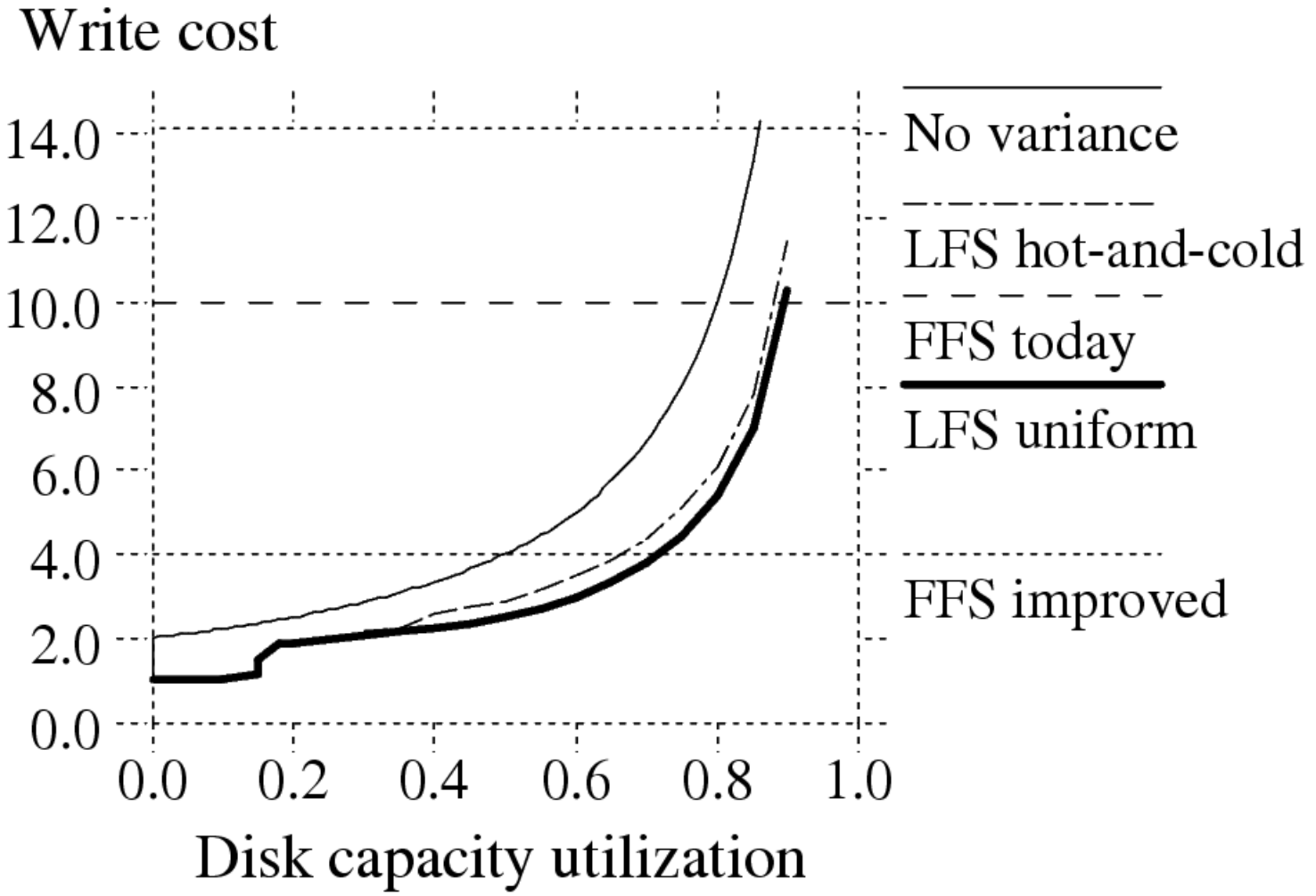
Simulation

- Goal: Measure the write cost with simple policies and two different access patterns.
  - File size of 4 KB.
  - Variable number of disk capacity.
  - The simulator overwrites one of the files with new data using one of two policies: Uniform and Hot-and-cold.

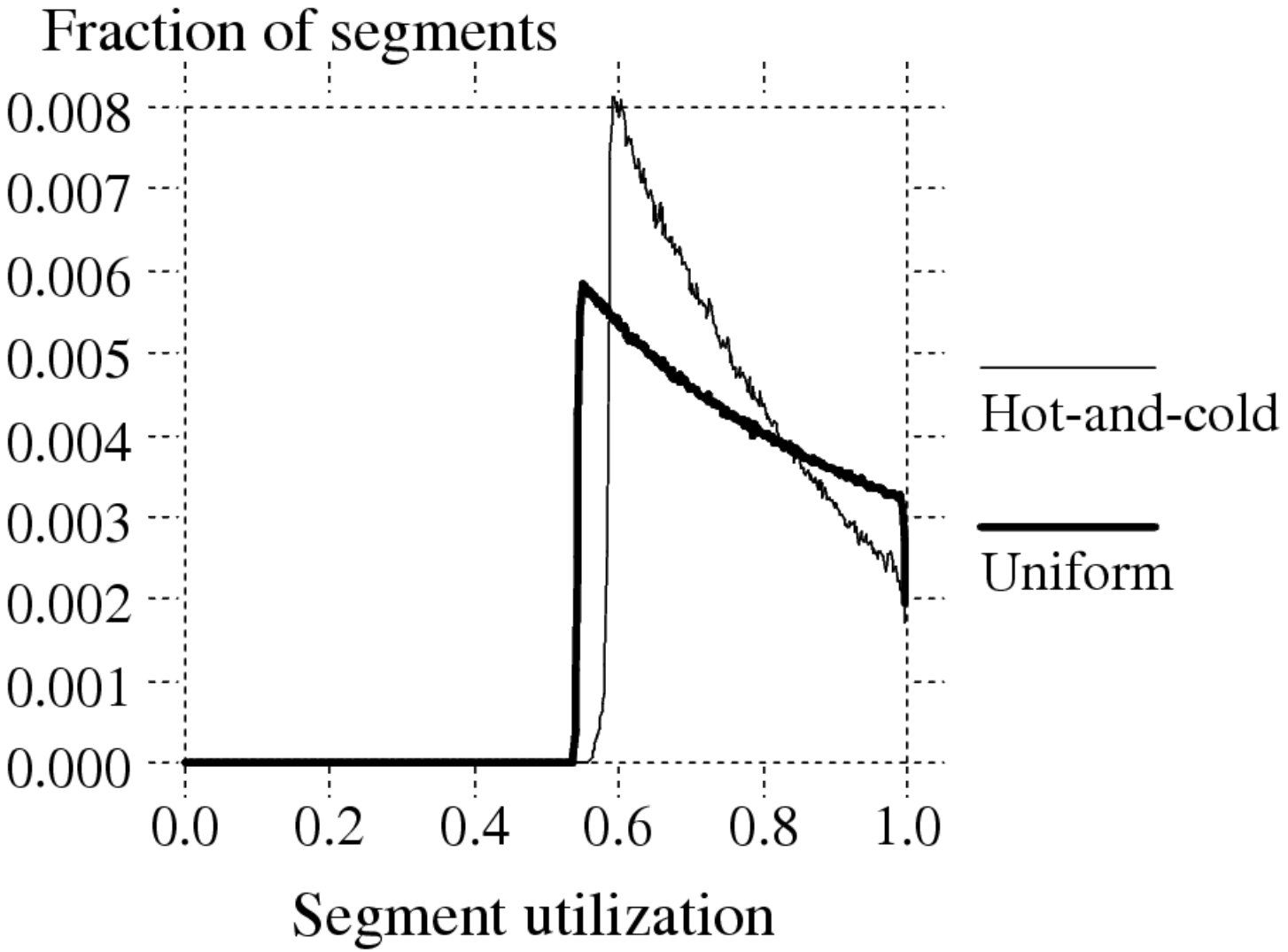
Simulation (cont)

- The simulator runs the cleaner when the clean segments are exhausted.
- Cleans until a fixed number of segments are clean.
- Chooses the least-utilized segment to clean.
- The simulator runs until the write cost is stabilized.

Write Costs in Simulation



Why does locality go bad?

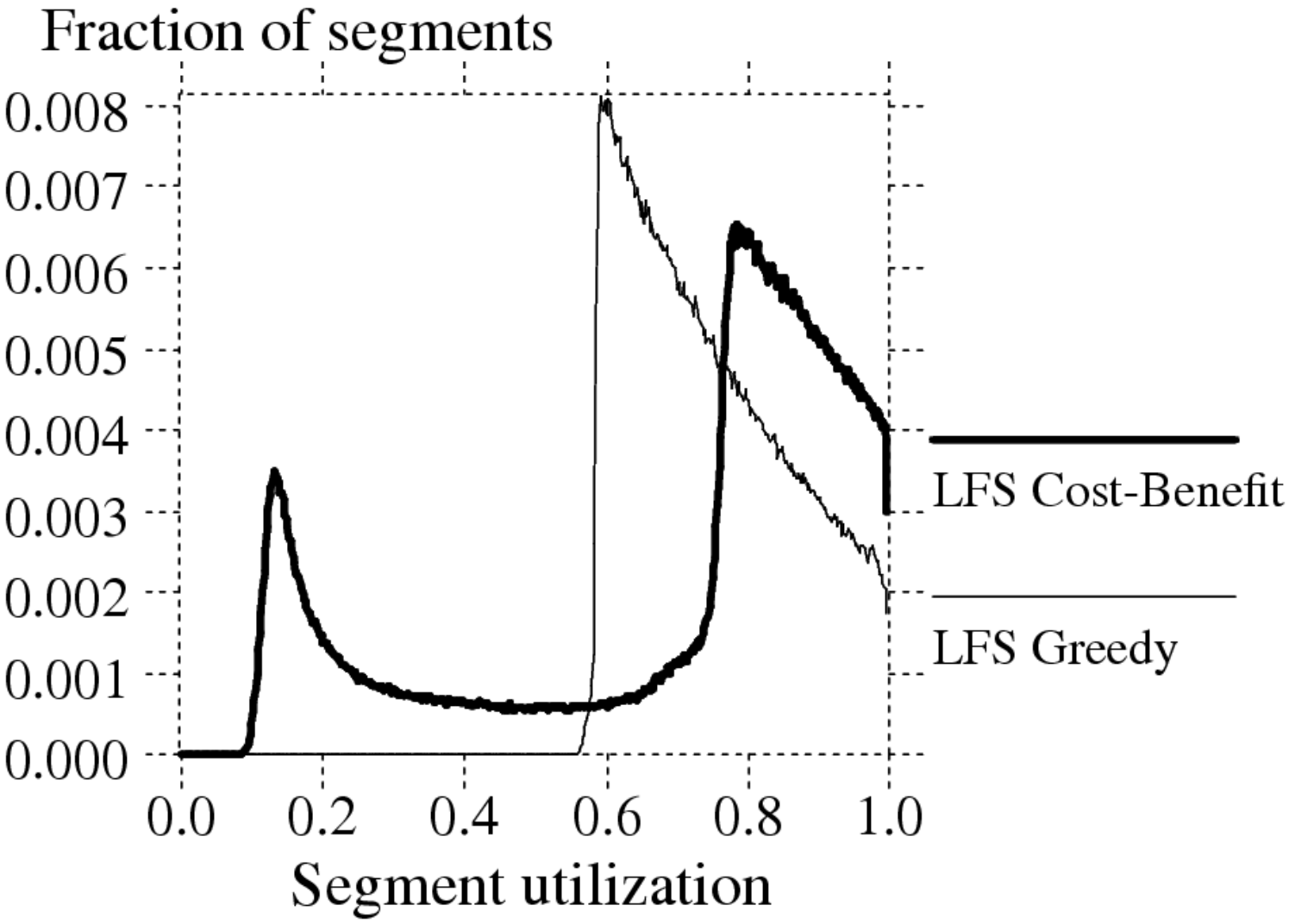


Solution: Cost-Benefit policy

- Goal: Give more value to free space in cold segments than in hot segments.

$$\frac{benefit}{cost} = \frac{free\ space\ generated\ \cdot age\ of\ data}{cost}$$
$$= \frac{(1-u)\cdot age}{1+u}$$

Cost-Benefit Results



Crash Recovery

- UFS must scan all the metadata structures to restore consistency.
- LFS has the most recent changes at the end of the log. Use two techniques:
  - Checkpoint: position in the log at which the file system is consistent and complete.
  - Roll-Forward: scanning of the LFS segments that were written after the last checkpoint.

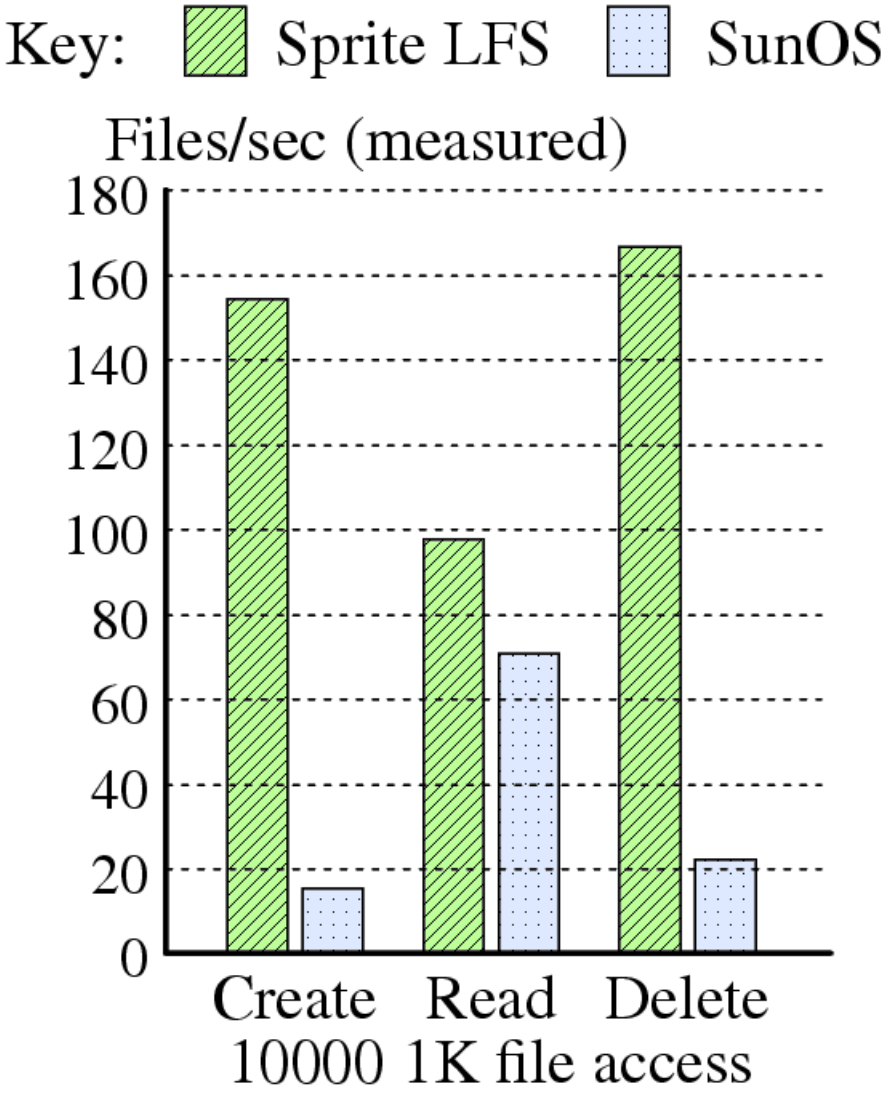
Implementation of LFS

- Implemented for UNIX-like OS Sprite.
- All the features implemented, but roll-forward was not installed in production.
- Simple implementation and re-use of some code.

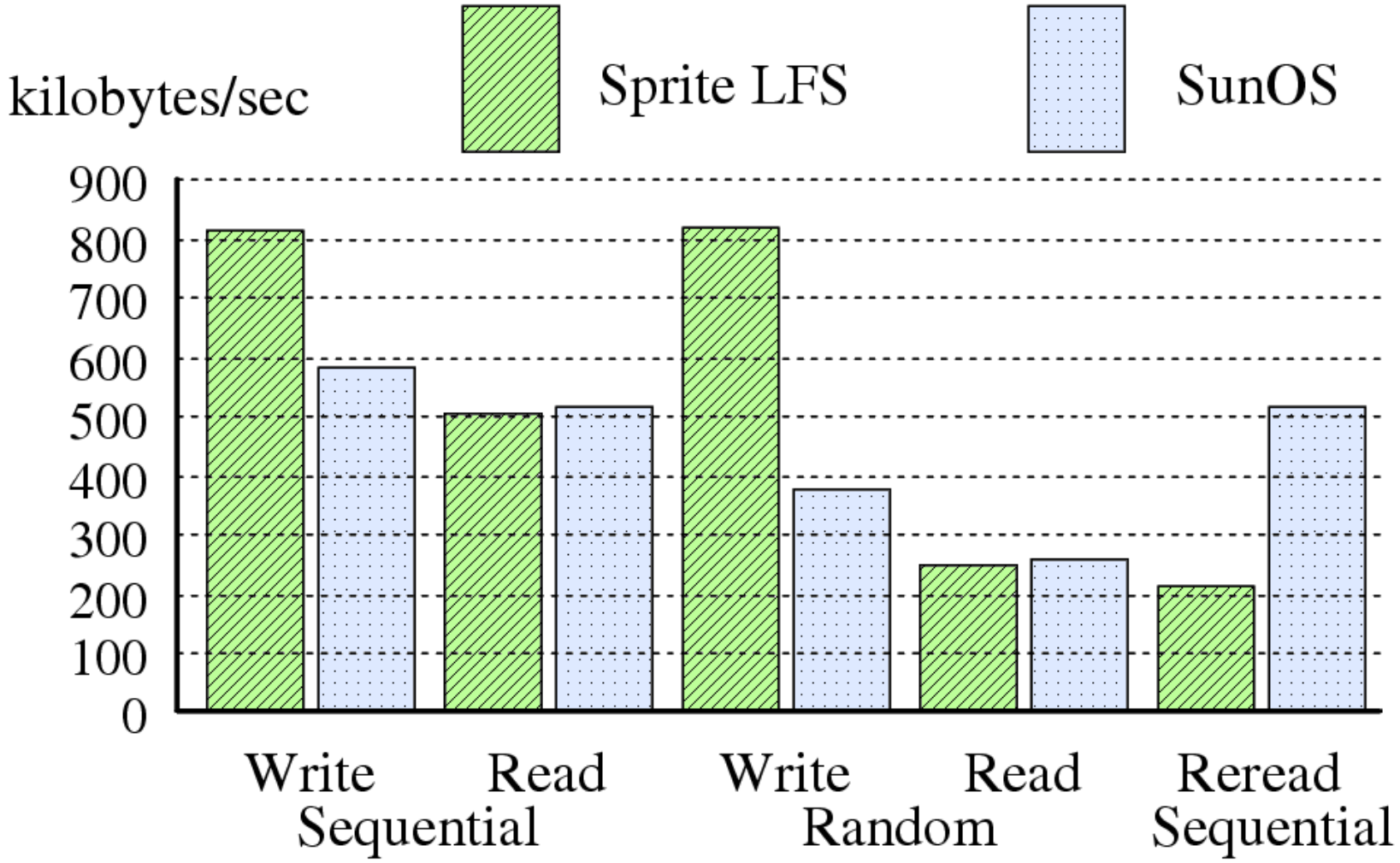
Experience with SpriteLFS

- Compared: SunOS 4.0.3 (UFS-like) and Sprite LFS.
- Benchmark 1: create, read and delete a large number of small files.
- Benchmark 2: create a 100MB file & perform some operations.
- Overhead not included

Micro-benchmark Results (small files)



Micro-benchmark Results (large file)



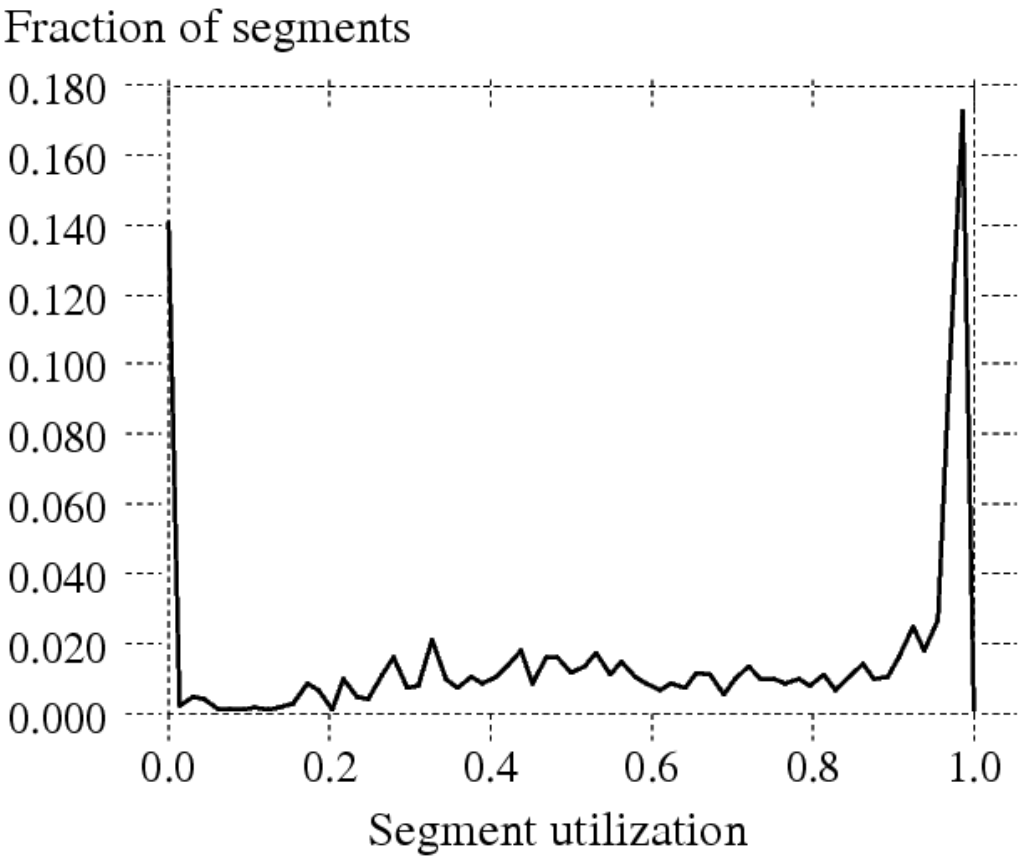
Cleaning Overheads

- Evaluation in a production log-structured file system over a four-month period.
- The file system /user6 was measured. Consist in the home directory of Sprite developers.

Cleaning Overhead Results

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

Segment Utilization in the /user6 file system



Conclusions

- Caching writes and making a single large I/O make extensive use of disk's bandwidth.
- Maintaining large extents of free space is difficult with low cleaning overhead. Solution: cost-benefit based cleaning policy.

Conclusions (cont)

- The Sprite LFS implementation works very well even for large files

References

- Mendel Rosenblum and John K. Ousterhout, “The Design and Implementation of a Log-Structured File System” *ACM Transactions on Computer Systems,* (1992)
- Mendel Rosenblum, “The Design and Implementation of a Log-structured File System”, *EECS Department, University of California*, Berkeley, (1992)
- Pradeep Padala, “A Log-Structured File System with Snapshots”, Summer of Code 2005, (2005)