The Design and Implementation of a Log-Structured File System

Phil Gibbons

15-712 F15

Lecture 10

Today's Reminders

- **Office Hours**
  - My office hours today 4:30-5:30 pm
- **Project Groups**
  - Should be formed by today
- **Friday's Class**
  - Two summaries for Friday
  - Special guest lecturer: Garth Gibson

The Design and Implementation of a Log-Structured File System
[SOSP'91, TOCS 1992]

- **Mendel Rosenblum** (Stanford, VMWare founder, ACM Doctoral Dissertation Award, Mark Weiser Award, ACM Systems Software Award, NAE)
- **John Ousterhout** (Stanford, IEEE Info Storage Sys Award, NAE, ACM Software System Award, ACM Grace Murray Hopper Award)

"The paper introduces log-structured file storage, where data is written sequentially to a log and continuously de-fragmented. The underlying ideas have influenced many modern file and storage systems like NetApp's WAFL file systems, Facebook's picture store, aspects of Google's BigTable, and the Flash translation layers found in SSDs."
– SigOps HoF citation

1992 Trends Prediction

- **Assumptions:**
  - CPU speeds increasing faster than disk speeds
  - Files are cached in main memory
  - Increasing memory sizes will make caches effective at satisfying read requests and buffering write requests
  - Crashes are infrequent & acceptable to lose seconds/minutes of work in each crash (use nonvolatile RAM if needed)
- **Thus, disk traffic will become dominated by writes**
- **Goal: "Design for file systems of the 1990s"**

Log-Structured File System

- **Log is the only structure on disk**
  - Avoids large overheads of disk seeks, important for short files
  - Contains indexing info for efficient reading
- **Most difficult challenge:**

  Need large extents of free space available for writing new data
  - Solution: Segments with a Segment Cleaner process

Problems with Existing File Systems

- **Spread info around the disk, causing many small accesses**
  - Berkeley UNIX FFS: 5 seeks + I/O to create a new file
  - <5% disk utilization
- **Write synchronously (esp. directories & inodes)**
  - Slow

File Location

- **inode map: maintains current location in log of each inode**
  - Include file version number
  - Keep active portions of inode map in memory

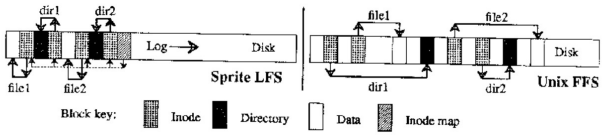
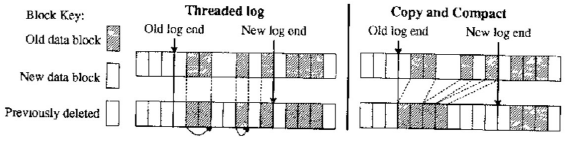


Fig. 1. A comparison between Sprite LFS and Unix FFS. This example shows the modified disk blocks written by Sprite LFS and Unix FFS when creating two single-block files named dir1/file1 and dir2/file2. Each system must write new data blocks and inodes for file1 and file2, plus new data blocks and inodes for the containing directories. Unix FFS requires ten nonsequential writes for the new information (the inodes for the new files are each written twice to ease recovery from crashes), while Sprite LFS performs the operations in a single large write. The same number of disk accesses will be required to read the files in the two systems. Sprite LFS also writes out new inode map blocks to record the new inode locations

Free Space Management: Segments

- **Threading free extents would cause severe fragmentation**
- **Copying is costly for long-lived files**



- **Segments**
  - Any given segment is written sequentially from beginning to end
  - Log is threaded on a segment-by-segment basis
  - Sprite LFS uses segments of size=512KB or 1 MB

Segment Cleaning Mechanism

- **Cleaning**
  - Read some segments into memory
  - Copy live data to a smaller number of clean segments
  - Reclaim original segments
- **Segment summary block: identifies each piece of info**
  - File number & block number for File data blocks
  - Uid = version number & inode number
- **Determine liveness by checking if file's inode or indirect block still refers to this block; otherwise block is dead**
  - No free-block list/bitmap, simplifying crash recovery

Segment Cleaning Policies

- **When: They run when # of clean segs < few 10s**
- **On how many: few 10s at a time until # clean > 50-100**
- **Which segments**
- **Grouping of live data**

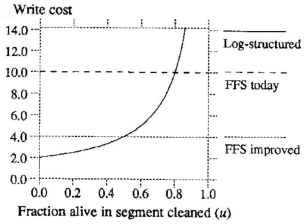
Primary factors determining LFS performance

- **Write cost: avg time disk is busy per byte of new data written, including all cleaning overhead**
  - E.g., write cost=10 means 10% of disk BW used for writing new data, 90% used for seeks, rotational latency, cleaning
  - With large segments, can ignore seeks & rotational latency

Analyzing Write Cost

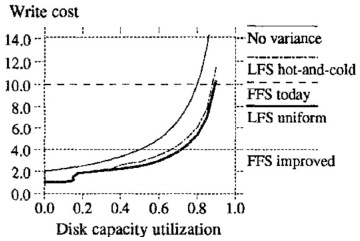
- **Read N segs, write out $N * \mu$ segments of live data**
  - μ is utilization of the segments, $0 \leq \mu < 1$

$$\text{write cost} = \frac{\text{total bytes read and written}}{\text{new data written}} = \frac{\text{read segs} + \text{write live} + \text{write new}}{\text{new data written}} = \frac{N + N*u + N*(1-u)}{N*(1-u)} = \frac{2}{1-u}$$



- **Trade-off**
  - LFS performance "can be improved by reducing the overall utilization of the disk space"

Impact of Locality: A Surprise!

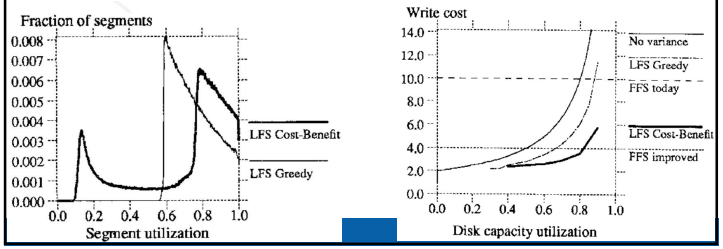


- **Simulator, 4KB files, select file to overwrite with new data**
  - Uniform: File selected uniformly at random
  - Hot-and-cold: 10% of files that are selected 90% of the time
  - Clean least utilized segment; Hot-and-cold writes out by age
- **Surprise: Locality & "better" grouping is worse! Why?**

Cost-Benefit Policy

- **Problem: Cold segments tie up many free blocks for long periods of time**
- **Solution: Factor in amount of time space likely to stay free**
  - Estimate as time since any block in segment was modified

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



Data Structures Stored on Disk

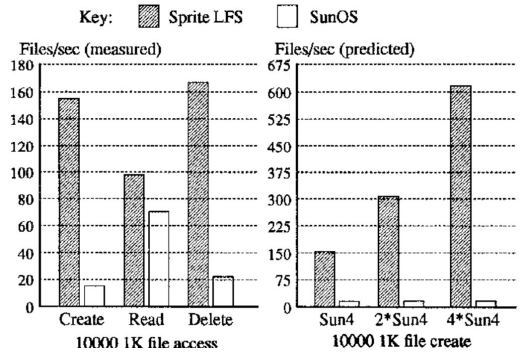
Data structure	Purpose	Location
Inode	Locates blocks of file, holds protection bits, modify time, etc.	Log
Inode map	Locates position of inode in log, holds time of last access plus version number.	Log
Indirect block	Locates blocks of large files.	Log
Segment summary	Identifies contents of segment (file number and offset for each block).	Log
★ Segment usage table	Counts live bytes still left in segments, stores last write time for data in segments.	Log
Superblock	Holds static configuration information such as number of segments and segment size.	Fixed
Checkpoint region	Locates blocks of inode map and segment usage table, identifies last checkpoint in log.	Fixed
Directory change log	Records directory operations to maintain consistency of reference counts in inodes.	Log

Crash Recovery

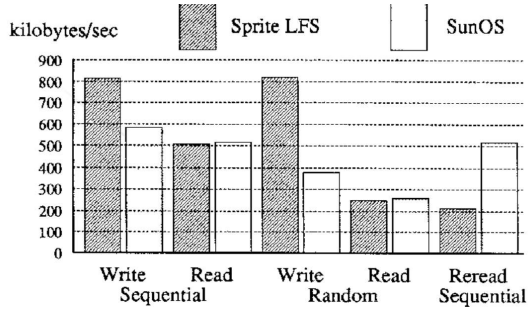
- **Checkpoints (every 30 secs – "much too short")**
  - Write all modified info to the log
  - Write a **checkpoint region** to fixed position on disk: addrs of all blocks in inode map & segment usage table, ptr to last segment written, current time
- **Roll-Forward from log**
  - New inode: update inode map
  - New data blocks w/o new inode: ignore
  - Adjust stats in segment usage table
  - LFS writes directory changes to log before corresponding directory block or inode (directory changes are forbidden during checkpoints)

Sprite LFS vs. SunOS (Unix FFS)

- **Sprite 4KB blocks & 1MB segments, SunOS 8KB blocks**



100MB Files, Different Access Patterns



- **Logical locality vs. Temporal locality**

Cleaning Overheads

Table II. Segment Cleaning Statistics and Write Costs for Production File Systems.

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

- **Sprite LFS is much better than simulation study predicts**
  - Large files that are written/deleted as a whole: greater locality
  - Cold segments much colder
  - Thus, highly bimodal segment utilization

Bottom Line

- **LFS can use disks an order of magnitude more efficiently**
  - Can use 70% of the disk bandwidth vs. 5-10% for Unix FFS

"This should make it possible to take advantage of several more generations of faster processors before I/O limitations once again threaten the scalability of computer systems."

How LFS Differs

- **From Garbage Collectors**
  - Sequential accesses necessary for high FS performance
  - Blocks belong to one file at a time: easier to identify garbage
- **From Databases**
  - Log is final repository for data, so must write entire blocks not deltas (compaction would hurt read performance)
  - Need cleaning to reclaim log space vs. delete on apply
  - Simpler crash recovery since don't need redo
- **From FTLs on SSDs?**

Friday's Papers

A Case for
Redundant Arrays of Inexpensive Disks (RAID)
David Patterson, Garth Gibson, Randy Katz
[Sigmod'88]

SigOps Hall of Fame paper

Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?
Bianca Schroeder, Garth Gibson
[Fast'07]

Fast'07 best paper