The Design and Implementation of a Log-Structured File System

Mendel Rosenblum and John K. Ousterhout

Outline

- Background
- Overview of Log-Structured File Systems
- Overview of Sprite LFS
- Where are they now?
- Critical Analysis
- References

Computing in the Early 90's

- Processor speed growing at an exponential rate
- Main memory increasing
- Disk performance (transfer bandwidth & access time) not improving relative to CPU speed
- Office environments dominated by writes to small files

File Systems in the early 90's

- Primary UNIX file system: FFS (Fast File System)
- Information spread around the disk causing to many small accesses
- Creating a new file requires at least five separate disk I/O's
- Synchronous writes
  - Application must wait for write to complete
  - Doesn't benefit from faster CPU's

Log-Structured File Systems

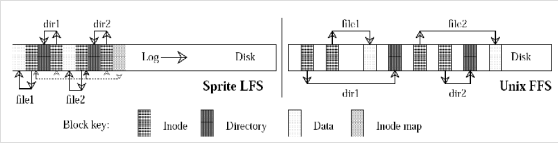
- Fundamental Idea
  - Buffer file system changes in a file cache
  - write all changes to the disk in a single operation
- Attempt to utilize as much raw disk bandwidth as possible
- Tend to have good write performance
- Easier to recover from crashes

Sprite LFS

- A prototype log-structured system
- Sprite network operating system
  - Research operating system
  - Developed at the University of California, Berkeley

Sprite LFS: Locating and Reading a File

- inodes
  - Similar structure to FFS
  - Written to the log
- inode map: data structure that maintains the current location of each inode
  - Able to cache in main memory



Sprite LFS: Free Space Management

- Most difficult design issue
- Free space is fragmented
- File system has two choices
  - Threading: leave data in place
    - Leads to fragmenting
    - Ends up no more efficient than FFS
  - Copying: Compact data
    - Can be costly
- Sprite LFS uses a combination of threading and copying

Sprite LFS: Segments

- Segments: fixed-sized extents of a disk
  - Always written to sequentially
  - Live-data must be moved out from a segment before the segment can be rewritten
- The log is treaded on a segment-by-segment basis
  - Long lived data does not have to be moved repeatedly

Sprite LFS: Segment Cleaning

- Three-step process
  (1) Read a number of segments into memory
  (2) Identify live data
  (3) Write live data back to a smaller number of clean segments
- Segment summary block identifies:
  - live blocks
  - the file to which each block belongs
  - the blocks position within a file

Sprite LFS: Segment Cleaning Cont.

- Policy issues:
  - When to clean?
  - How many segments to clean?
  - What segments should be cleaned?
  - How to group live blocks while cleaning?
- Use threshold values to answer first two
- Last two questions are more complicated

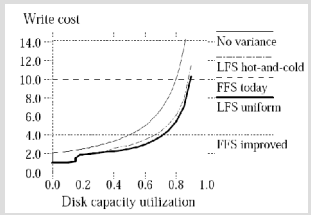
Write Cost

- Write cost: the average amount of time the disk is busy per byte of new data written
- Used to compare cleaning policies
- A write cost of 1.0 is perfect

$$\text{write cost} = \frac{\text{total bytes read and written}}{\text{new data written}}$$

Uniform vs Hot-and-Cold

- Uniform: All files have an equal chance of being selected
- Hot-and-Cold: 10% of the files are selected 90% of the time



Sprite LFS: Segment Management

- "Hot" and "cold" segments should be manged differently
- More important to clean cold segments
- Hot segments are more likely to die quickly
- Segments "temperature" based upon age of data
  - Segment usage table

Log Structured File Systems Today

- Didn't catch on
- NILFS: a LFS implementation for Linux
- Sometimes used on storage media (flash drives, CD-RW's, etc)
  - Limit amount of writes
- Journaling file system

Further Research

M. Seltzer: "An Implementation of a Log-Structured File System for UNIX", 1993

A critique of the above paper by John Ousterhout: http://www.eecs.harvard.edu/margo/papers/usenix95-lfs/supplement/

Analysis

- Authors provided a detailed analysis of what worked, what didn't, and why not
- Benchmarking
- Simulation results

References

[1] M. Seltzer: “An Implementation of a Log-Structured File System for UNIX”, 1993
[2] J. Ousterhout: “The Sprite Network Operating System”, 1988
[3] J. Matthews: “Improving the performance of log-structured file systems with adaptive methods”, 1997