

# 1 Log Structured File System

## Vocabularies

**Question** Are the diagrams of LFS on chapter 43, the diagram in memory segment or hard drive?

### 1. Log-Structured File System

- Is a file system that buffers all updates including metadata to a circular buffer.
- When full, is written to a disk in a one long sequential transfer to an unused part of the disk

### 2. Segment

- Means a storage of fixed size to put the large chunk of updates, so writes can be performed at one time

### 3. Metadata

- Means data about data
- It is used to summarize basic information about data which can make tracking and working with specific data easier

## Example

- Time and date of creation
- Creator or author of data
- Number of free blocks in hard drive
- File Size

### 4. Inode

- Is a short form for **index node**
- Has a low-level name called **i-number**
- Contains all the information you need about a file (i.e. metadata)

### 5. Write Buffering

- Is a region of a physical memory storage used to temporarily store data before writing to disk

### 6. Amortization

- Means the action or process of gradually writing off the initial cost of an asset.

- In this context, means the more you write, the better, and closer you get to achieving peak bandwidth.

## 7. Inode map

- Is a table indicating where each inode is on the disk



## 8. Overhead

- Is any combination of excess or indirect resources that are required to perform a specific task

## 9. Checkpoint Region

- Is a fixed location on disk storing pointer to latest pieces of imap



## 10. Recursive Update Problem

- Is the problem concerned with a file system never updating in place but to new locations on disk

## 11. Garbage

- Is old versions of file structures scattered throughout the disk after write

## 12. Clean

- Means removal of old dead versions of file data, inodes, and other structures to make blocks on disk free again for use in subsequent writes

## 13. Garbage Collection

- Is a technique that arises in programming languages that automatically free unused memory for programs

**14. Hole**

- Is an unallocated block between allocated blocks in disk space

**15. Compaction**

- Refers to combining all the empty spaces together.

**16. Segment Summary Block\***

- I need to come back on this one
- Is a block that contains a pointer to the next summary block to link segments into one long chain that LFS treats as a linear log

**17. Roll Forward\***

- I need to come back on this one

**18. Shadow Paging**

- Is the process of writing to an unused portion of the disk, and then reclaim the old space through cleaning

## 1.1 Log-Structured File Systems

- Motivations

**1. System memories are growing**

- Exploits memory getting bigger year after year
- Serve reads through cache → disk traffic is increasingly consists of writes
- File performance now determined write performance

**2. There is a large gap between random I/O performance and sequential I/O performance**

- Hard-drive transfer bandwidth increased over the years
  - \* Due to more more bits packed to surface of a drive
- Hard drive's seek and rotation delay decreased slowly
  - \* Hard to cheaply make motor that runs faster
- Sequential read gives sizeable performance advantage than causing seeks and rotations

**3. Existing file systems perform poorly on many common workloads**

- FFS
  - \* Requires a large number of writes to create a new file with one new block
  - \* Incurs many short seeks and subsequent rotational delays and performance falls

**4. File systems are not raid aware**

- Buffers all updates in an in-memory **segment**
- Writes to disk only when buffer is full as a long sequential transfer to unused part of disk

## 1.2 Writing to Disk Sequentially

- Place data block first
- Place inode next to data block with its pointers pointing to the data block
  - Works the same as inodes in UNIX (e.g ext 2, ext4)



## 1.3 Writing Sequentially And Effectively

- Writing a block as it comes adds rotational delay
  - Write a block + wait + write second block
  - Not good at performance
- A large number of contiguous writes required for peak performance
  - Done using **Write buffer**
    - \* Put all updates in an in-memory **segment**
    - \* When full, write the segment all at once to the disk
      - Writes are efficient when segment is large enough

## 1.4 How much to Buffer

- Depends on the disk

## 1.5 Problem: Finding Inodes

- Inodes are scattered throughout the disk
- Data blocks and inodes are not fixed
  - On each update, data blocks and inode are placed on new sequential blocks
  - Need to know where they are :(

- Solution: The inode map + checkpoint region
  - Inode map
    - \* Takes an inode number as input
    - \* Produces the disk address of the most recent version of inode
    - \* Is read after checkpoint region
    - \* Is cached of its entirety
      - All imaps read and placed in memory
    - \* Fixes the problem of finding moving inodes
    - \* Problem: inode map is not fixed
  - Checkpoint region
    - \* Fixes the problem of moving inode map
    - \* Contains pointers to the latest piece of the inode map
    - \* Is fixed
    - \* Is read first

## 1.6 Reading A File from Disk: A Recap

- Checkpoint region is read first
- Inode maps are read second
  - All are cached in memory
- Inodes are read third
- Data blocks are read last

## 1.7 What About Directories

- Is identical to UNIX file systems
  - Imap points to directory inode
  - Directory inode points to directory block
  - Directory block has entries containing user-readable name and its i-number

### Example

`("user-readable-name", i-number)`

- Imap solves **recursive update problem**
  - \* Changes are not written in directory
    - If it did, a small update in inode would have caused changes upto root
  - \* Imap has the information about latest changes

## 1.8 New Problem: Garbage Collection

- LFS repeatedly writes the latest version of a file to new locations on disk
- LFS also keeps the old version of file
- If used as a feature → **versioning file system**
- If to be removed → use **garbage collection**
  - Works on **segment** by segment basis
    - \* Periodically reads old segments including live segments
    - \* Write a new segment containing only live blocks
      - Number of new blocks  $N$  are smaller than old blocks  $M$
      - The process is called **compaction**
    - \* Free up old ones for writing
  - No removal by individual basis
    - \* External fragmentation
    - \* Results in drop in performance
- Question #1 (mechanism): How to tell which blocks in a segment is live, and which is dead?
- Question #2 (policy): How often should cleaner run? which segments should it pick to clean?

## 1.9 Determining Block Liveness

- Answers the question "How to tell which blocks in a segment is live, and which is dead?"
- Is known via **segment summary block**
  - Stores
    - \* File number
    - \* Block number of file data blocks
    - \* Inode number
  - Determines liveness by checking if file's inode or indirect block still refers to this block; otherwise block is dead

## 1.10 A Policy Question: Which Blocks To Clean, And When?

- Answers the question "Which blocks to clean, and when?"
  - Cold segment first, hot segment second
    - \* hot segment's contents frequently change
    - \* cold segment's contents doesn't change often

## 1.11 A Crash Recovery and the Log

- Answers the question: "What happens if the system crashes while LFS is writing to disk?"
    - Two cases to consider
      - 1.
      2. Ensuring CR updates regularly
- \*