

A Survey of File Systems

Steve Martin
Dave Latham

Copyright note: figures and some elements from slides by Hank Levy, CSE451, University of Washington. Thank you, Hank!

Unix

- Original (i.e. 1970's) Unix FS was very simple and straightforwardly implemented
 - Too many long seeks
 - No one uses this anymore
- BSD UNIX folks did a redesign in the mid '80's
 - FFS: "Fast File System"
 - Improved disk utilization, decreased response time
 - Basic idea is FFS is aware of disk structure
 - i.e., place related things on nearby cylinders to

Unix FS

- Original (non-FFS) unix FS had two major problems:
 - 1. data blocks are allocated randomly in aging file systems
 - blocks for the same file allocated sequentially when FS is new
 - as FS "ages" and fills, need to allocate blocks freed up when other files are deleted
 - problem: deleted files are essentially randomly placed
 - so, blocks for new files become scattered across the disk!
 - 2. inodes are allocated far from blocks
 - all inodes at beginning of disk, far from data
 - traversing file name paths, manipulating files, directories requires going back and forth from inodes to data blocks
- Also, small blocks (1KB) caused two problems:
 - low bandwidth utilization
 - small max file size (function of block size)
- Performance was unacceptable before long.

Unix FFS

- FFS addressed these problems using notion of a cylinder group
 - disk partitioned into groups of cylinders
 - data blocks from a file all placed in same cylinder group
 - files in same directory placed in same cylinder group
 - inode for file in same cylinder group as file's data
- Introduces a free space requirement
 - to be able to allocate according to cylinder group, the disk must have free space scattered across all cylinders
 - in FFS, 10% of the disk is reserved just for this purpose!
- FFS uses a larger block (4KB)
 - allows for very large files (1MB only uses 2 level indirect)
 - but, introduces internal fragmentation
 - there are many small files (i.e., <4KB)
 - fix: introduce "fragments"
 - 1KB pieces of a block

Log-Structured File System (LFS)

- LFS was designed in response to two trends in workload and disk technology:
 - 1. Disk bandwidth scaling significantly (40% a year)
 - but, latency is not
 - 2. Large main memories in machines
 - therefore, large buffer caches
 - absorb large fraction of read requests in caches
 - can use for writes as well
 - coalesce small writes into large writes
- LFS takes advantage of both to increase FS performance
- Solves some FFS problems:
 - FFS: placement improved, but can still have many small seeks
 - possibly related files are physically separated
 - inodes separated from files (small seeks)
 - directory entries separate from inodes

LSF (2)

- The basic idea: treat the entire disk as a single log for appending
 - collect writes in the disk buffer cache, and write out the entire collection of writes in one large request
 - leverages disk bandwidth with large sequential write
 - no seeks at all! (assuming head at end of log)
 - all info written to disk is appended to log
 - data blocks, attributes, inodes, directories, etc.
- There are two main challenges with LFS:
 - 1. locating data written in the log
 - FFS places files in a well-known location, LFS writes data "at the end of the log"
 - 2. managing free space on the disk
 - disk is finite, and therefore log must be finite
 - cannot always append to log!
 - need to recover deleted blocks in old part of log

LSF (3)

- How LFS locates data
 - LFS appends inodes to end of log, just like data
 - makes them hard to find
 - So, we use another level of indirection: inode maps
 - inode maps map file #s to inode location, kept in a checkpoint region
 - Checkpoint region has a fixed location
 - Cache inode maps in memory for performance
- LFS: append-only quickly eats up all disk space!
 - need to recover deleted blocks
 - fragment log into segments and thread on disk
 - segments can be anywhere
 - reclaim space by cleaning segments
 - read segment
 - copy live data to end of log
 - now have free segment you can reuse!
 - cleaning is a big problem
 - costly overhead, when do you do it?

Windows FAT

- "The" Windows FS up until NTFS became prevalent.
 - Two flavors:
 - FAT16 (older, 2.1 GB file size limit)
 - FAT32 (2 TB file size limit, new features)
- Basically, here's how it works:
 - File Allocation Table (yes, actual table) at the top of the volume.
 - Two copies kept in case one becomes damaged.
 - FAT location and root directory stored in fixed location for boot.
 - Disk is allocated in 'clusters', the size of which is determined by the size of the volume.
 - Why do you think this is?
 - On file creation, entry is created in directory file as well as in FAT.
 - Files are given first open location in drive.
 - If more than one cluster in file, entry in FAT points to next cluster.
- Advantages:
 - Simple!
- Disadvantages:
 - Slow, dangerous to have all volume information in one place, inefficient use of disk space, as size of volume increases performance decreases, etc.

Windows NTFS

- Windows Network File System
 - Introduced with Windows NT and the default used for Windows XP.
- Very complicated file system
 - There are whole books on this, check web for more info.
- Here are the absolute basics. . .
 - Still uses the concept of clusters, size of which range from 512 B to 64 KB
 - For each file created, a record is created in the Master File Table.
 - This is not a FAT!
 - Does not contain cluster addresses, contains file attributes.
 - Record is used to locate possibly scattered clusters of file.
 - ~12.5% of the disk is reserved for MFT
 - NTFS attempts to lay out all files contiguously.
 - However, in addition to MFT, each cluster also records data about the next cluster in the file.
 - Directories are stored as B-trees, with lazy deletion!
 - What does this mean?
 - Operations take the form of transactions against the file system.
 - This allows for guarantees to be made for data consistency, improving file