W4118: FFS and LFS

Instructor: Junfeng Yang

References: Modern Operating Systems (3rd edition), Operating Systems Concepts (8th edition), previous W4118, and OS at MIT, Stanford, and UWisc

File system examples

- BSD Fast File System (FFS)
  - What were the problems with Unix FS?
  - How did FFS solve these problems?

- Log-Structured File system (LFS)
  - What was the motivation of LFS?
  - How did LFS work?

Original Unix FS

- From Bell Labs
- Simple and elegant

Unix disk layout

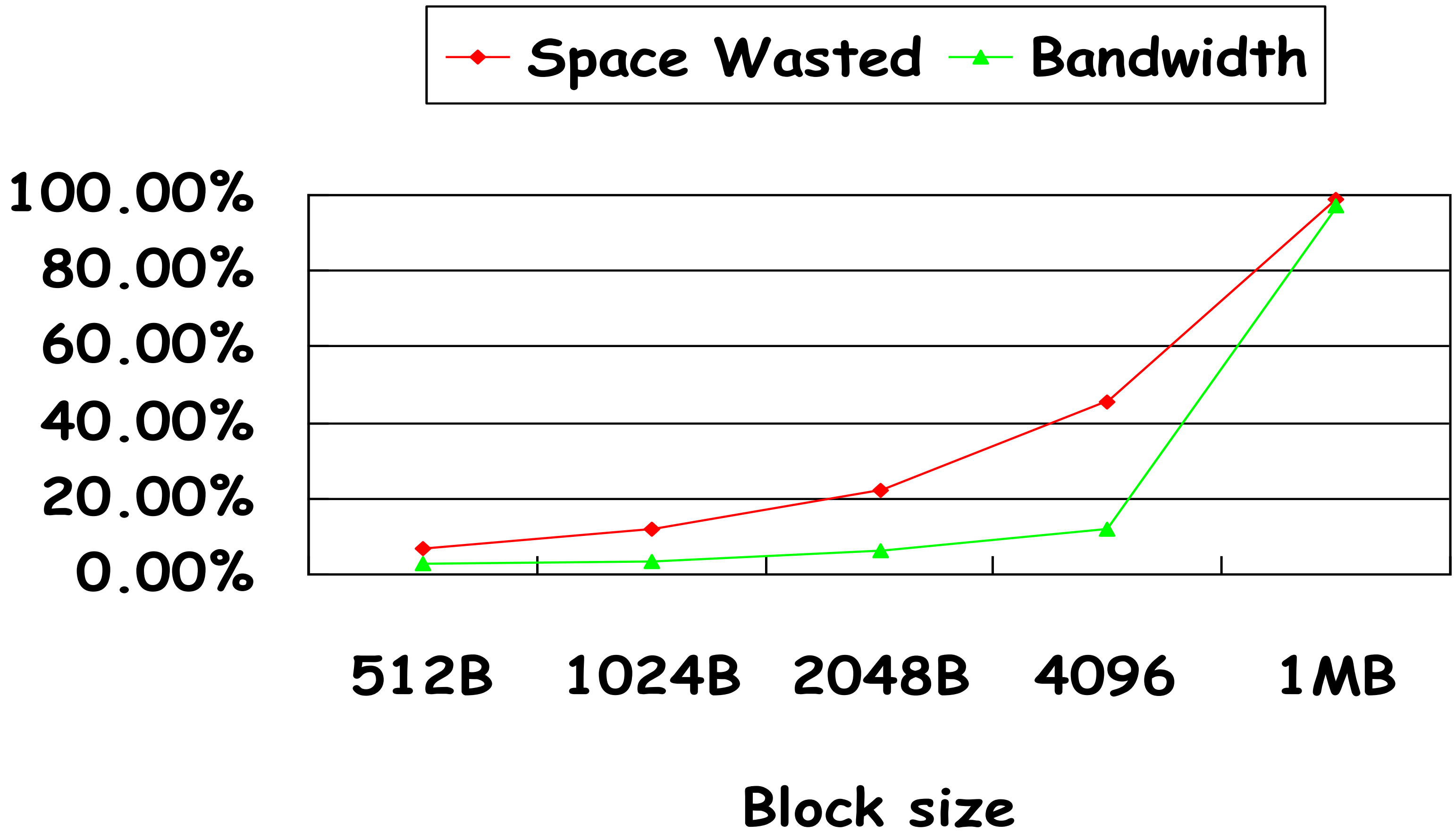
super	inodes	data blocks (512 bytes)

- Problem: slow
  - 2% of maximum disk bandwidth even for sequential disk transfer (20KB/s)

Why so slow?

- Problem 1: blocks too small
  - Fixed costs per transfer (seek and rotational delays)
  - Require more indirect blocks
- Problem 2: unorganized freelist
  - Consecutive file blocks are not close together
  - Pay seek cost even for sequential access
- Problem 3: no data locality
  - inodes far from data blocks
  - inodes of files in directory not close together

Problem 1: blocks too small

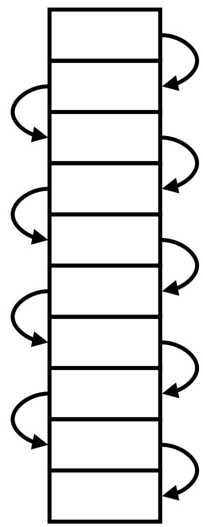


Larger blocks

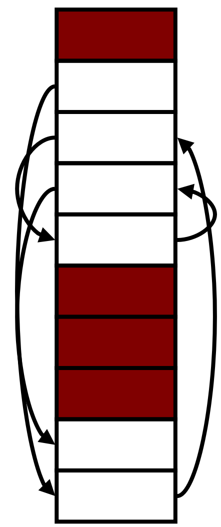
- BSD FFS: make block 4096 or 8192 bytes
- Solve the internal fragmentation problem by chopping large blocks into small ones called fragments
  - Algorithm to ensure fragments only used for end of file
  - Limit number of fragments per block to 2, 4, or 8
  - Keep track of free fragments
- Pros
  - High transfer speed for larger files
  - Low wasted space for small files or ends of files
- This internal fragmentation problem is not a big deal today

Problem 2: unorganized freelist

- Leads to random allocation of sequential file blocks overtime



Initial performance good



Get worse over time

Fixing the unorganized free list

- Periodical compact/defragment disk
  - Cons: locks up disk bandwidth during operation
- Keep adjacent free blocks together on freelist
  - Cons: costly to maintain
- Bitmap of free blocks
  - Bitmap: each bit indicates whether block is free
    - E.g., 010001000101010000001
    - cache (all or parts of) bitmap in mem ➔ few disk ops
  - Used in BSD FFS

Problem 3: data Locality

- Locality techniques
  - Store related data together
  - Spread unrelated data apart
    - Make room for related data
  - Always find free blocks nearby
    - Rule of thumb: keep some free space on disks (10%)

- FFS new organization: cylinder group
  - Set of adjacent cylinders
  - Fast seek between cylinders in same group
  - Each cylinder group contains superblock, inodes, bitmap of free blocks, usage summary for block allocation, data blocks

Achieving locality in FFS

- Maintain locality of each file
  - Allocate data blocks within a cylinder group
- Maintain locality of inodes in a directory
  - Allocate inodes in same dir in a cylinder group
- Make room for locality within a directory
  - Spread out directories to cylinder groups
  - Switch to a different cylinder group for large files

BSD FFS performance improvements

- Achieve 20-40% of disk bandwidth on large files
  - 10X improvements over original Unix FS
  - Stable over FS lifetime
  - Can be further improved with additional placement techniques

- Better small file performance

- More enhancements

File system examples

- BSD Fast File System (FFS)
  - What were the problems with Unix FS?
  - How did FFS solve these problems?
- Log-Structured File system (LFS)
  - What was the motivation of LFS?
  - How did LFS work?

Log-structured file system

- Motivation
  - Faster CPUs: I/O becomes more and more of a bottleneck
  - More memory: file cache is effective for reads
  - Implication: writes compose most of disk traffic

- Problems with previous FS
  - Perform many small writes
    - Good performance on large, sequential writes, but many writes are still small, random
  - Synchronous operation to avoid data loss
  - Depends upon knowledge of disk geometry

LFS idea

- Insight: treat disk like a tape-drive
  - Best performance from disk for sequential access
- Write data to disk in a sequential log
  - Delay all write operations
  - Write metadata and data for all files intermixed in one operation
  - Do not overwrite old data on disk

Pros and cons

- Pros
  - Always Large sequential writes → good performance
  - No knowledge of disk geometry
    - Assume sequential better than random
- Potential problems
  - How do you find data to read?
  - What happens when you fill up the disk?

Read in LFS

- Same basic structures as Unix
  - Directories, inodes, indirect blocks, data blocks
  - Reading data block implies finding the file's inode
    - Unix: inodes kept in array
    - LFS: inodes move around on disk
- Solution: inode map indicates where each inode is stored
  - Small enough to keep in memory
  - inode map written to log with everything else
  - Periodically written to known checkpoint location on disk for crash recovery

Disk cleaning

- When disk runs low on free space
  - Run a disk cleaning process
  - Compacts live information to contiguous blocks of disk

- Problem: long-lived data repeatedly copied over time
  - Solution: partition disk in to segments
    - Group older files into same segment
    - Do not clean segments with old files

- Try to run cleaner when disk is not being used

- LFS: neat idea, influential
  - Paper on LFS is likely the most widely cited OS paper
  - Real file systems based on the idea