CIS452 Lecture-40 Notes

Winter 2017

Scribe: *Tyler Paquet*

Announcements

- Continued to look through handout on Block Allocation

Chapter 11 (Continued)

1. Indexed Allocation: Limitations
   - The maximum file size if the number of pointers that can fit into one index block
   - To store large files
     - Link several index blocks together
     - Multi-level index
     - Combined: use both direct index and multilevel indices (Unix File Systems)
2. UFS File Format
   - UFS (Unix File System) refers a set of FS variants based on Unix design
   - Boot block (usually a single sector): code to bootstrap the OS
   - Super block (several disk sectors): metadata about the filesystem
   - Inode block (several disk sectors) for storing index nodes (inodes)
     - One inode per file/directory
   - Many data blocks (greater than 95 percent of the disk sectors)
     - Directory entry and File objects
     - Journals used during recovery
3. UFS Super Block
   - Size of the entire filesystem
   - Information on free data blocks
     - Number of free blocks
     - A list of free blocks
   - Information on free inodes

    - Number of free inodes
    - A list of free inodes
    - The inode for the root (/) directory (inode 128 = root)

4. Linux/Unix File Allocation
    - One inode per file / directory
    - File block addresses are stored in inode and also in indirect blocks
    - N (10 or 12) direct pointers and 3 indirect pointers
        - N direct pointers hold the addresses of the first N data blocks used by a file
        - The N+1st pointer holds the address of a single indirect block
        - The N+2nd pointer holds the address of a double indirect block
        - The N+3rd pointer holds the address of a triple indirect block
5. Indirect Blocks
    - Indirect blocks are data blocks used for storing pointers to data blocks
    - Single indirect blocks
        - Contain pointers to data blocks
    - Double indirect blocks
        - Contain pointers to single indirect blocks
    - Triple indirect blocks
        - Contain pointers to double indirect blocks
6. Free (Disk) Block Management
    - Bit Vector/Bitmap
    - Linked-List (Free List)
        - Simple
        - Grouping
        - Counting
    - In the list can be implemented as a tree (for faster search)
7. Bit Vector / Bitmap
    - Each block is represented by 1-bit (0 = free, 1= used)
    - Number of bytes for bitmap = number of total disk blocks
        - A disk of 64 blocks =¿ 64 bits (8 bytes) needed for bitmap
    - Fast to locate free blocks: use bitwise operations
    - Require extra space to maintain free blocks (tooooo much wasted space)
        - A disk with capacity 512G (= 239) and block size of 512 bytes (= 29
        - Number of blocks = 239/29 = 230 blocks

    - Size of bitmaps 230/23 = 227 bytes (128M) of bitmap
    - Percentage of wasted space = 227 / 239 = 2-12 = 1/4096 2.5

8. Linked-List of Free Blocks / Free List

    - Improvements over Bitmap technique
        - Chain all the free blocks together
        - The superblock keeps the head and tail of this list
    - Place the next pointer inside (at the end) of the freeblocks
    - Searching for contiguous free blocks requires reading each block in the chain (too many disk I/O)

9. Efficiency of Disk Block Operations

    - Any bytes in RAM can be accessed within the same amount of time (ns)
    - But, blocks on a disk may have different access time depending on their relative distance to the R/W head current position
    - Updating just one byte on a disk block requires writing the entire block
    - Design objectives of various FS are
        - higher storage capacity (how to support Petabytes or more)
        - fewer disk operations
        - highly consistent data (fewer recovery required)

10. Critical On-Disk Data Structures

    - A filesystem essentially holds (at least) three separate trees:
        - A tree of files and directories (user-owned)
        - A tree of free data blocks (maintained by OS)
        - A tree of free index blocks (maintained by OS)
    - These trees originate from the superblock!
    - Any updates to the FS must guarantee the three trees are in sync!

11. Low-Level Filesystem Operations

    - To write a file into a (Linux) filesystem
        - Write the file contents into the data blocks
        - Update the inode (file size, timestamps, block pointers, .)
        - Update the superblock (update the free-block list free-inode list)
    - When the above sequence does not run to completion (i.e. by power failure), the filesystem records only a partial (inconsistent) data/metadata of your file

12. When a new file is created

    - Its data blocks are allocated [superblock update]

- One inode is allocated [superblock update and inode update]
- The directory entry for the file is update [data block update]
- The inode of the containing directory is updated [inode update

13. Database Transactions (Intent) Logs

    - To guarantee data integrity
    - Multiple operations that update different parts of the DB are performed under one transaction: BEGIN TRANSACTION and COMMIT/ROLLBACK
      - In addition to changing the data, a transaction also logs the intended changes (insert, delete, update)
      - The log must be securely saved PRIOR TO the modification of the actual data
    - At recovery time, the contents of the log are compared with the actual data and unfinished transactions can be recovered

14. Journaling / Log-Structured FS

    - Every modification to the FS must be associated with a log entry
      - Log entry format: timestamp, action, blockid, oldvalue, newValue
      - Log entries are created for ALL types of modification (including inode and superblock)
    - The log must be saved FIRST before the actual block updates take place
    - Delete the log after the block updates are successful
    - Recovery after system crash: use the existing log entries to restore FS