CS 455: INTRODUCTION TO DISTRIBUTED SYSTEMS [FILE SYSTEMS]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- How many choices in CAP?
- Routing paths in regular graphs?
- After a random identifier has been generated, how is this new node communicated to its neighbors?

Topics covered in this lecture

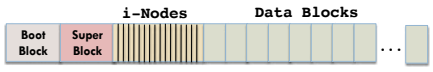
- File Systems
  - UFS inodes
- Distributed File Systems
  - Naming
  - Replication
  - Semantics of file sharing

Several file systems are in use

- CD-ROMs written in ISO 9660 format
  - Designed by CD manufacturers
- UNIX
  - Unix file system (**UFS**)
  - Berkley Fast File System (**FFS**)
- Windows: **FAT, FAT32** and **NTFS**
- Linux
  - Supports 40 different file systems
  - Extended file system: **ext2, ext3** and **ext4**

INODES

Disk Layout in traditional UNIX systems



An integral number of inodes fit in a single data block

Super Block describes the state of the file system

- Total size of the partition
- Block size and number of disk blocks
- Number of inodes
- List of free blocks
- inode number of the root directory

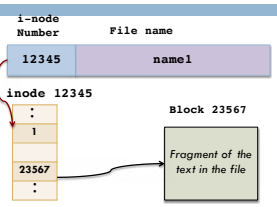
- Destruction of super block?
  - Will render file system unreadable!

A linear array of inodes follows the super block

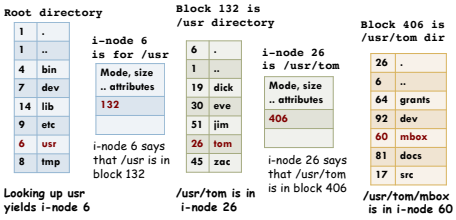
- inodes are numbered from **1** to some **max**
- Each inode is identified by its inode number
  - inode number contains info needed to **locate** inode on the disk

- Users think of files as filenames
  - UNIX thinks of files in terms of inodes

Directory entry, inode and data block for a simple file



Looking up path names in UNIX Example: /usr/tom/mbox



Advantages of directory entries that have name and inode information

- Changing filename only requires changing the directory entry
- Only 1 physical copy of file needs to be on disk
  - File may have several names (or the same name) in different directories
- Directory entries are small
  - Most file info is kept in the inode

inode

- **Fixed-length** data structure
  - One per file
- Contains information about
  - File **attributes**
    - Size, owner, creation/modification time etc.
  - **Disk addresses**
    - File blocks that comprise file

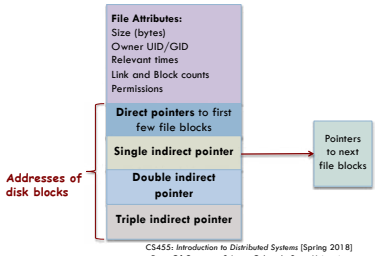
inode

- The inode is used to encapsulate information about a large number of file blocks
- For e.g. {Block size = 8 KB, and file size = 8 GB}
  - There would be a million file-blocks
    - inode will store info about the **pointers to these blocks**
  - inode allows us to access info for all these blocks
    - *Storing pointers to these file blocks also takes up storage*

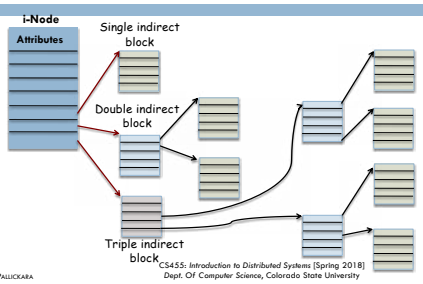
Managing information about data blocks in the inode

- First few data blocks of the file stored in inode
- If the file is large: **Indirect** pointer
  - To a block of pointers that point to additional data blocks
- If the file is larger: **Double indirect** pointer
  - Pointer to a block of indirect pointers
- If the file is huge: **Triple indirect** pointer
  - Pointer to a block of double-indirect pointers

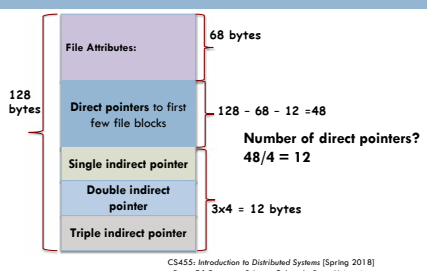
Schematic structure of the inode



inode: How the pointers to the file blocks are organized



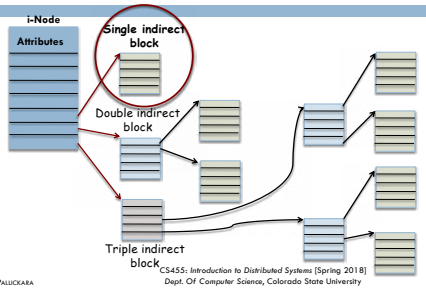
inode: A quantitative look
BLOCK Size = 8 KB and Pointers = 4 bytes



inode: A quantitative look
BLOCK Size = 8 KB and Pointers = 4 bytes

- 12 **direct** pointers to file blocks
- Each file block = 8KB
- Size of file that can be represented with direct pointers
  - 12 x 8 KB = 96 KB

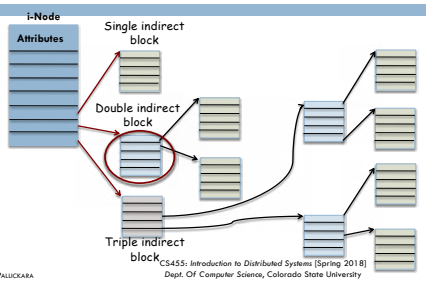
inode



inode: A quantitative look
BLOCK Size = 8 KB and Pointers = 4 bytes

- Block size = 8 KB
- Single indirect block =block size = 8 KB (8192 bytes)
- Number of pointers held in a single-indirect-block
  - Block-size/Pointer-size
  - 8192/4 = 2048
- With single-indirect pointer
  - Additional $2048 \times 8KB = 2^{11} \times 2^3 \times 2^{10} = 2^{24}$ of a file can be addressed

inode



inode: A quantitative look
BLOCK Size = 8 KB and Pointers = 4 bytes

- With a **double indirect pointer** in the inode
  - The double-indirect block has 2048 pointers
    - Each pointer points to a different single-indirect-block
    - So, there are 2048 single-indirect blocks
  - Each single-indirect block has 2048 pointers to file blocks
- Double indirect addressing allows the file to have an additional size of
  - $2048 \times 2048 \times 8\ KB = 2^{11} \times 2^{24} = 2^{35}$ …. (32 GB)

inode: A quantitative look
BLOCK Size = 8 KB and Pointers = 4 bytes

- **Triple indirect addressing**
  - Triple indirect block points to 2048 double indirect blocks
  - Each double indirect block points to 2048 single indirect block
  - Each single direct block points to 2048 file blocks
- Allows the file to have an additional size of
  - $2048 \times 2048 \times 2048 \times 8\ KB = 2^{11} \times 2^{35} = 2^{46}$ (64 TB)

Limits of triple indirect addressing

- In our example:
  - There can be 2048 x 2048 x 2048 data blocks
  - *i.e.*, $2^{11} \times 2^{11} \times 2^{11} = 2^{33}$
  - Pointers would need to be longer than 32-bits to fully address this storage

What if we increase the size of the pointers to 64-bits (data block is still 8 KB) ?

- What is the maximum size of the file that we can hold?
- 8 KB data block can hold (8192/8) = 1024 pointers
- **Single indirect** can add
  - $1024 \times 8\ KB = 2^{10} \times 2^3 \times 2^{10} = 2^{23}$ (8MB) of additional bytes to the file

What if we increase the size of the pointers to 64-bits (data block is still 8 KB)?

- **Double indirect** addressing allows the file to have an additional size of
  - $1024 \times 1024 \times 8\ KB = 2^{10} \times 2^{23} = 2^{33}$ …. (8 GB)
- **Triple indirect** addressing allows the file to have an additional size of
  - $1024 \times 1024 \times 1024 \times 8\ KB = 2^{10} \times 2^{33} = 2^{43}$ (8 TB)

DISTRIBUTED FILE SYSTEMS

Distributed File Systems (DFS) allow sharing of physically dispersed files

- File service **activity** has to be carried *over the network*
- Distinctive features
  - Multiplicity
  - Autonomy
  - Dispersion

Clients, Servers & Storage devices

The look-and-feel of a distributed file system

- Appears as a **conventional, centralized** system
- DFS client interface should
  - **Not distinguish** between local and remote files
  - **Locate** files and arrange for **transport** of data
- Facilitate **user mobility**
  - Bring user's environment wherever the user logs in

Most important performance measure: TIME needed to service requests

- Conventional systems
  - Disk-access time
  - (small) CPU-processing time
- DFS **adds remote access**
  - Deliver request to server
  - Get response across the network to the client
  - CPU overhead involving communication software

NAMING IN DISTRIBUTED FILE SYSTEMS

Naming is a mapping between logical and physical addresses

- Users deal with filenames
  - System manipulates disk-blocks on disk tracks
- **Multilevel** mapping provides a file abstraction
  - Hides *how* and *where* the files are stored
- User specifies filenames
  - Mapped to a numerical identifier
    - Mapped to disk blocks

} Multilevel mapping

Range of mapping filenames in traditional and distributed file systems

- Traditional
  - Address within a disk
- DFS
  - Include machine where the file is stored

DFS adds a new dimension to the abstraction …

- **Hide** where (in the network) the file is located
- One step away is file replication
- Given a filename, the mapping returns
  - Location of the file's replicas
- Existence of multiple copies and their locations are **hidden**

Naming structure and physical storage location

- File name does not reveal storage location
  - **Location transparency**
- File name does not change when storage location changes
  - **Location independence**
  - Dynamic mapping
- **File migration**
  - Changing the location of a file automatically

Once there is a separation of name and location …

- Clients can access files residing on servers
- Clients can also be **diskless**
  - Rely on servers for all files
    - Including the OS
- Current trend is to use {local + remote} storage
  - LOCAL: OS and networking software
  - REMOTE: User data and applications

Naming Schemes in Distributed File Systems

- Combination of host-name and local-name
  - Guarantees system-wide unique name
    - Not location transparent or independent
- NFS
  - Attach remote directories to local directories
    - Appearance of coherent directory structure
- Single global name structure
  - Spans all files in the system

Remote File Access: Once the server has been located, actual data transfer must take place

- Remote-service mechanism
  - Requests delivered to server
  - Server performs accesses
  - Results delivered back to the client
  - **RPC-based** approaches implement this
- Remote-service analogous to
  - Performing disk-access for each access in conventional file systems

Semantics of File Sharing

When two users share the same file

- Define **semantics** of reading & writing precisely
- Avoid problems

Semantics of file sharing on single processor systems

- When READ follows WRITE
  - READ returns the value that was just written
- When READ follows two successive WRITEs
  - READ returns value that was written last
- Absolute time-ordering on all operations
  - Returns most recent value
  - **UNIX semantics**

Achieving UNIX semantics in a distributed system

- Possible if there is only **one server** AND
  - Clients **do not cache** files
  - READs and WRITEs processed strictly sequentially
- Performance is quite **poor**, so …
  - Allow clients to maintain local copies of files
    - In their private caches

But what if a client locally modifies the cache, and another reads same file from server?

- **Session semantics**
  - Changes to an open file are visible only to process that modified it
  - Only when file is ***closed*** is the change made visible to other processes

Make all files immutable

- No way to open a file for writing
- Only operations on a file are
  - CREATE and READ
- What happens if two processes replace the same file at the same time?
  - **Solution:** Allow one of the new files to replace the old one

The transactional approach

- Access to a file (or a group of files) done in a **transactional** fashion
- System guarantees that calls within transaction
  - Will be carried out **in order**
  - **Without interference** from other concurrent transactions

Contrasting approaches to dealing with shared files in distributed systems

Method	Comment
UNIX semantics	EVERY operation on a file is **instantly visible** to all processes
Session semantics	No changes are visible to other processes **until the file is closed**
Immutable files	**No updates** are possible Simplifies sharing and replication
Transactions	All changes have the **all-or-nothing** property

The contents of this slide set are based on the following references

- Avi Silberschatz, Peter Galvin, Greg Gagne. *Operating Systems Concepts, 8th edition. Publisher - John Wiley & Sons, Inc. ISBN-13: 978-0-470-12872-5.* [Chapter 10]
- *Andrew S Tanenbaum. Modern Operating Systems. (3rd Edition, 2007). Publisher - Prentice Hall. ISBN: 0136006639/978-0136006633.* [Chapter 4]
- Kay Robbins & Steve Robbins. *Unix Systems Programming, 2nd edition, Publisher: Prentice Hall. ISBN-13: 978-0-13-042411-2.* [Chapter 5]