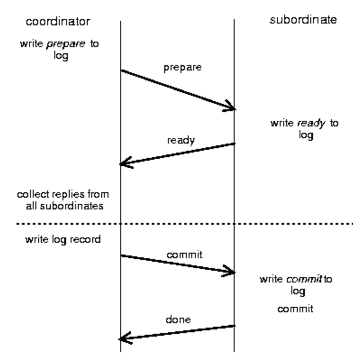
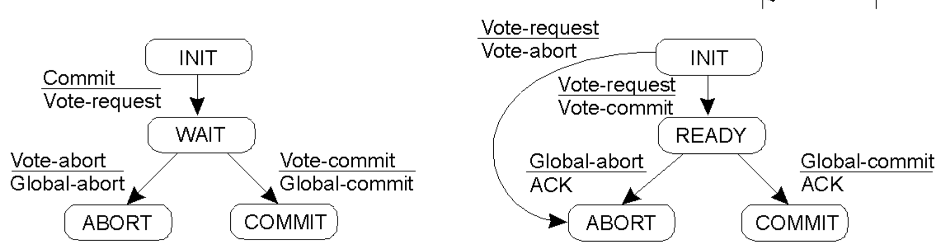
Last Class: Fault tolerance

- Reliable communication
  - One-one communication
  - One-many communication
- Distributed commit
  - Two phase commit
  - Three phase commit
- Failure recovery
  - Checkpointing
  - Message logging

Two Phase Commit

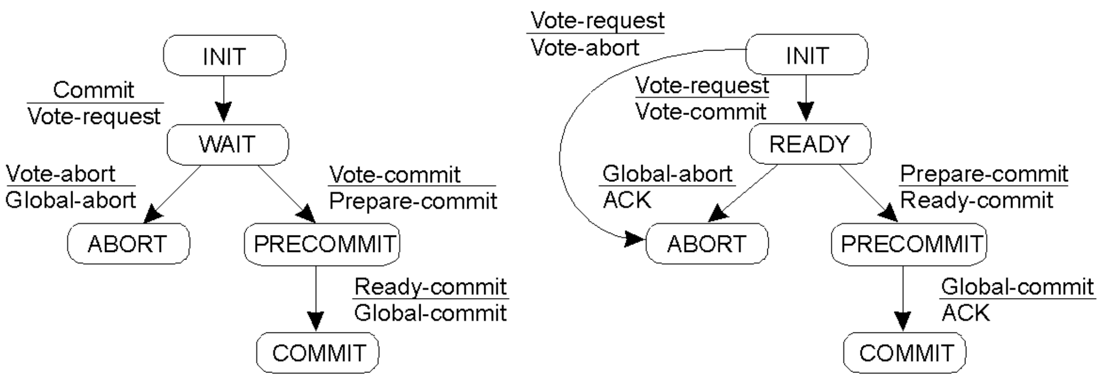
- Coordinator process coordinates the operation
- Involves two phases
  - Voting phase: processes vote on whether to commit
  - Decision phase: actually commit or abort





(a) (b)

Three-Phase Commit



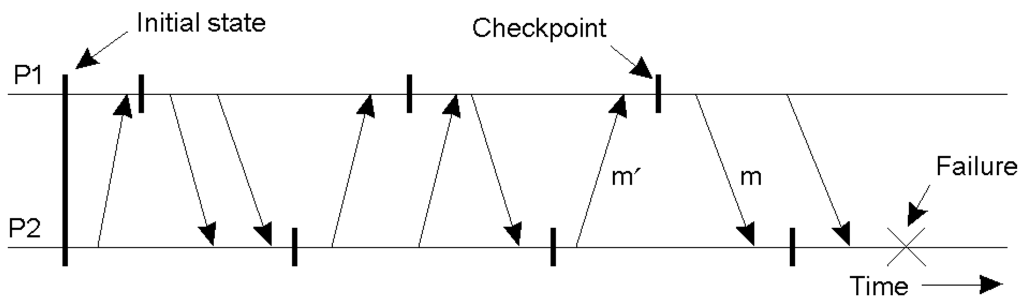
(a) (b)

Two phase commit: problem if coordinator crashes (processes block)
Three phase commit: variant of 2PC that avoids blocking

Recovery

- Techniques thus far allow failure handling
- Recovery: operations that must be performed after a failure to recover to a correct state
- Techniques:
  - Checkpointing:
    - Periodically checkpoint state
    - Upon a crash roll back to a previous checkpoint with a *consistent state*

Independent Checkpointing



- Each processes periodically checkpoints independently of other processes
- Upon a failure, work backwards to locate a consistent cut
- Problem: if most recent checkpoints form inconsistent cut, will need to keep rolling back until a consistent cut is found
- Cascading rollbacks can lead to a domino effect.

Coordinated Checkpointing

- Take a distributed snapshot [discussed in Lec. 11]
- Upon a failure, roll back to the latest snapshot
  - All process restart from the latest snapshot

Message Logging

- Checkpointing is expensive
  - All processes restart from previous consistent cut
  - Taking a snapshot is expensive
  - Infrequent snapshots => all computations after previous snapshot will need to be redone [wasteful]
- Combine checkpointing (expensive) with message logging (cheap)
  - Take infrequent checkpoints
  - Log all messages between checkpoints to local stable storage
  - To recover: simply replay messages from previous checkpoint
    - Avoids recomputations from previous checkpoint

Today: Distributed File Systems

- Overview of stand-alone (UNIX) file systems
- Issues in distributed file systems
- Next two classes: case studies of distributed file systems
    - NFS
    - Code
    - xFS
    - Log-structured file systems (time permitting)

File System Basics

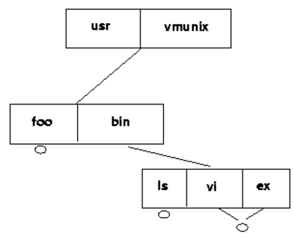
- File: named collection of logically related data
  – Unix file: an uninterpreted sequence of bytes
- File system:
  – Provides a logical view of data and storage functions
  – User-friendly interface
  – Provides facility to create, modify, organize, and delete files
  – Provides sharing among users in a controlled manner
  – Provides protection

File Types and Attributes

- File types
  – Directory, regular file
  – Character special file: used for serial I/O
  – Block special file: used to model disks [buffered I/O]
  – Strongly v/s weakly typed files
- File attributes: varies from OS to OS
  – Name, type, location, size, protection info, password, owner, creator, time and date of creation, last modification, access
- File operations:
  – Create, delete, open, close, read, write, append, get/set attributes

Directories

- Tree structure organization most common



- Access to a file specified by *absolute file name*
- User can assign a directory as the *current working directory*
  - Access to files can be specified by *relative name* relative to the current directory
- Possible organizations: linear list of files, hash table

Unix File System Review

- User file: linear array of bytes. No records, no file types
- Directory: special file not directly writable by user
- File structure: directed acyclic graph [directories may not be shared, files may be shared *(why?)*]
- Directory entry for each file
  - File name
  - inode number
  - Major device number
  - Minor device number
- All inodes are stored at a special location on disk [super block]
  - Inodes store file attributes and a multi-level index that has a list of disk block locations for the file

Inode Structure

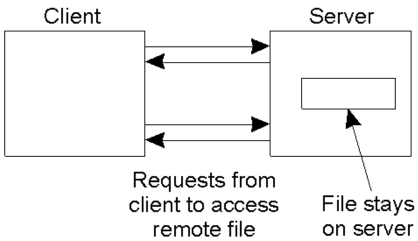
- Fields
  - Mode
  - Owner_ID, group_id
  - Dir_file
  - Protection bits
  - Last access time, last write time, last inode time
  - Size, no of blocks
  - Ref_cnt
  - Address[0], … address[14]
    - Multi-level index: 12 direct blocks, one single, double, and triple indirect blocks

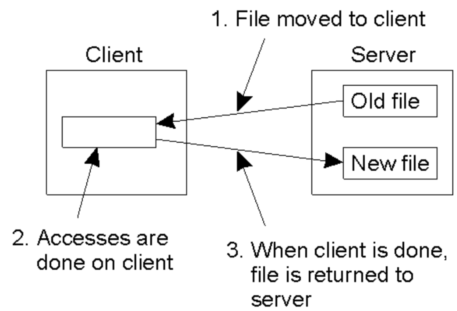
Distributed File Systems

- *File service:* specification of what the file system offers
  - Client primitives, application programming interface (API)
- *File server:* process that implements file service
  - Can have several servers on one machine (UNIX, DOS,…)

- Components of interest
  - File service
  - Directory service

File Service



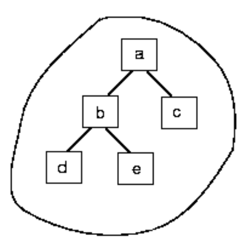


- Remote access model
  - Work done at the server
- Stateful server (e.g., databases)
- Consistent sharing (+)
- Server may be a bottleneck (-)
- Need for communication (-)

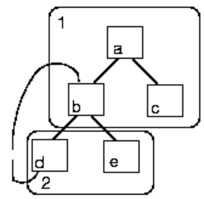
- Upload/download mode
  - Work done at the client
- Stateless server
- Simple functionality (+)
- Moves files/blocks, need storage (-)

Directory Service

- Create/delete files
- Hierarchical directory structure



- Arbitrary graph



System Structure: Server Type

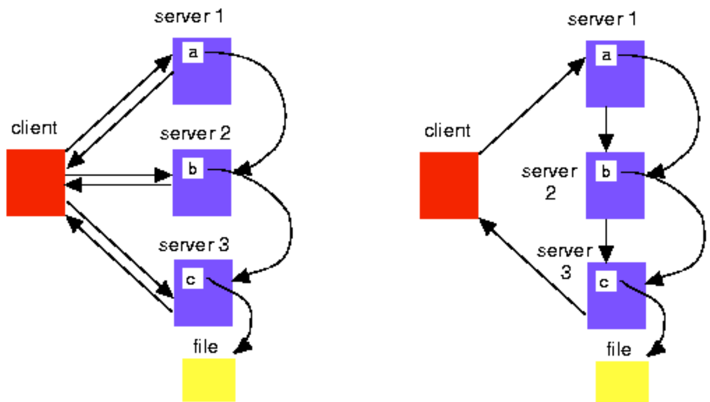
- Stateless server
  - No information is kept at server between client requests
  - All information needed to service a requests must be provided by the client with each request *(what info?)*
  - More tolerant to server crashes
- Stateful server
  - Server maintains information about client accesses
  - Shorted request messages
  - Better performance
  - Idempotency easier
  - Consistency is easier to achieve

System Structure

- Client v/s server implementations possibilities
  - Same process implements both functionality
  - Different processes, same machine
  - Different machines (a machine can either be client or server)
- Directory/file service – same server?
  - Different server processes: cleaner, more flexible, more overhead
  - Same server: just the opposite

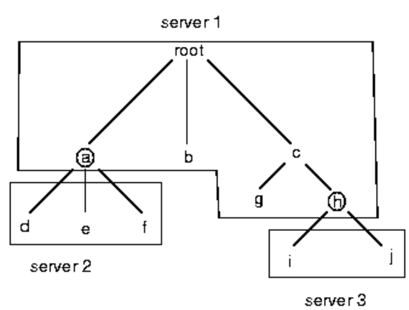
Naming Issues

- Path name lookup can be iterative or recursive
  - /usr/freya/bin/netscape



Naming Issues: Mounting

- Mounting: file system can be mounted to a node of the directory



- Depending on the actual mounts, different clients see different view of the distributed file system

File Sharing Semantics

- Unix semantics
  - Read after write returns value written
    - System enforces absolute time ordering on all operations
    - Always returns most recent value
    - Changes immediately visible to all processes
    - Difficult to enforce in distributed file systems unless all access occur at server (with no client caching)
- Session semantics
  - Local changes only visible to process that opened file
  - File close => changes made visible to all processes
  - Allows local caching of file at client
  - Two nearly simultaneous file closes => one overwrites other?

Other File Sharing Semantics

- Immutable files
  - Create/delete only; no modifications allowed
  - Delete file in use by another process
- Atomic transactions
  - Access to files protected by transactions
  - Serializable access
  - Costly to implement