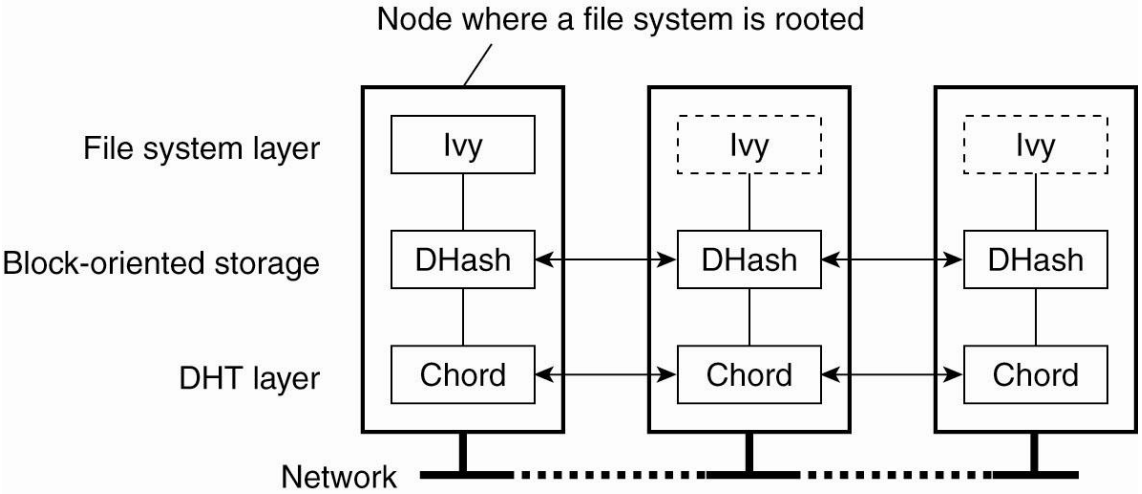
CS 550:
Advanced Operating Systems

Networked File Systems
Part 2

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
March 31st, 2011

Symmetric Architectures Distributed File Systems



- Commonly used DHT-based systems for distributing data, combined with a key-based lookup mechanism
- Key difference: whether build a file system on top of a distributed storage layer
- Example: Ivy

System Structure

- How should the system be organized?
  - Are clients and server different?
    - Same process implements both functionality
    - Different processes, same machine
    - Different machines (a machine can either be client or server)
  - How are file and directory services organized-same server?
    - Different server processes: cleaner, more flexible, more overhead
    - Same server: just the opposite
  - Caching/no caching
    - server
    - client
  - how are updates handled?
  - File sharing semantics?
  - Server type: stateful vs. stateless

System Structure: Server Type

- Stateless server
  - No info is kept about a request after the request is served
  - Request has to be self contained (e.g. contain complete file names)
  - Pro & Con?
  - Longer messages required
  - Mapping required for each request
  - Better tolerance to server crashes
  - No table required => no limit on the number of clients
  - File locking not possible => requires a locking server

System Structure: Server Type

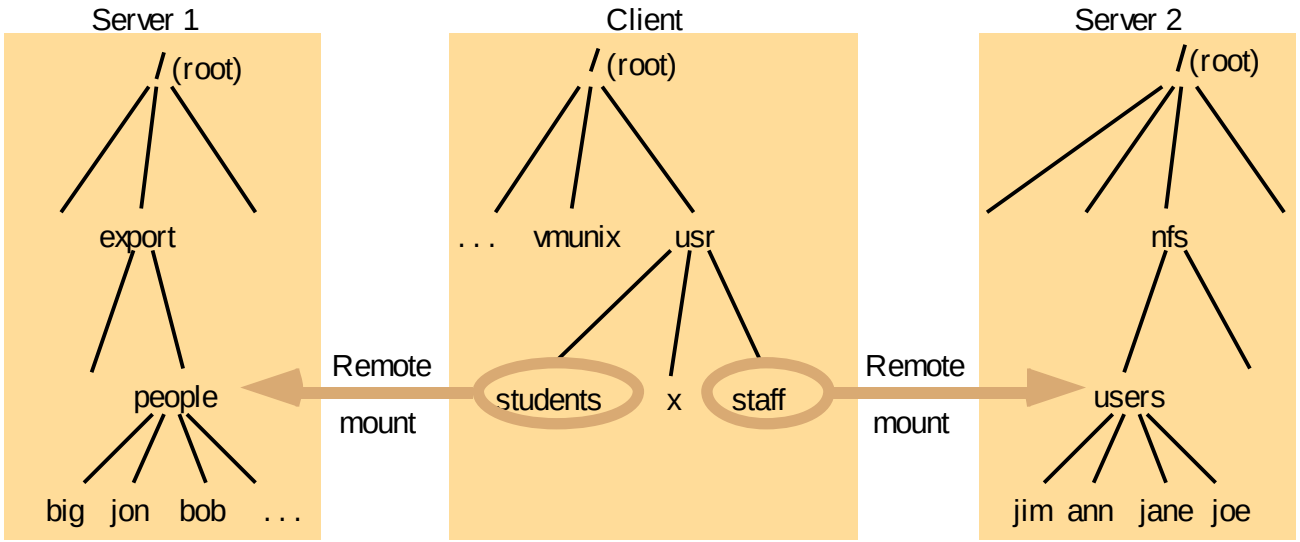
- Stateful server
  - Info kept about each client that has an open file
    - Server maintains information about client accesses
  - Shorted request messages
    - Request to open file needs complete file name => server returns file descriptor
    - Other request need file descriptor only => mapping not required for each request
  - Pro & Con?
  - Difficult recovery from server crashes
  - Server is the performance bottleneck
  - Better performance
  - Consistency is easier to achieve

Naming Issues

- Location transparency: the path name does not reveal the file location
  - E.g. /serverA/dir1/dir2/x does not say where the server is located
- Location independency: files can be moved and all references to them continue to be valid
  - E.g. /serverA/dir1/dir2/x is not location independent

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

Semantics of File Sharing

- Unix semantics: used in centralized systems
  - 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
- Issues in distributed systems
  - Single file server (no client caching): easy to implement UNIX semantics
  - Client file caching: improves performance by decreasing demand at the server; updates to the cached file are not seen by other clients
  - Conclusion:?
    - Difficult to enforce in distributed file systems unless all access occur at server (with no client caching)

Semantics of File Sharing

- Session semantics (relaxed 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
- Problems:
  - What if two or more clients are caching and modifying a file?
    - Final result depends on who closes last
    - Use an arbitrary rule to decide who wins

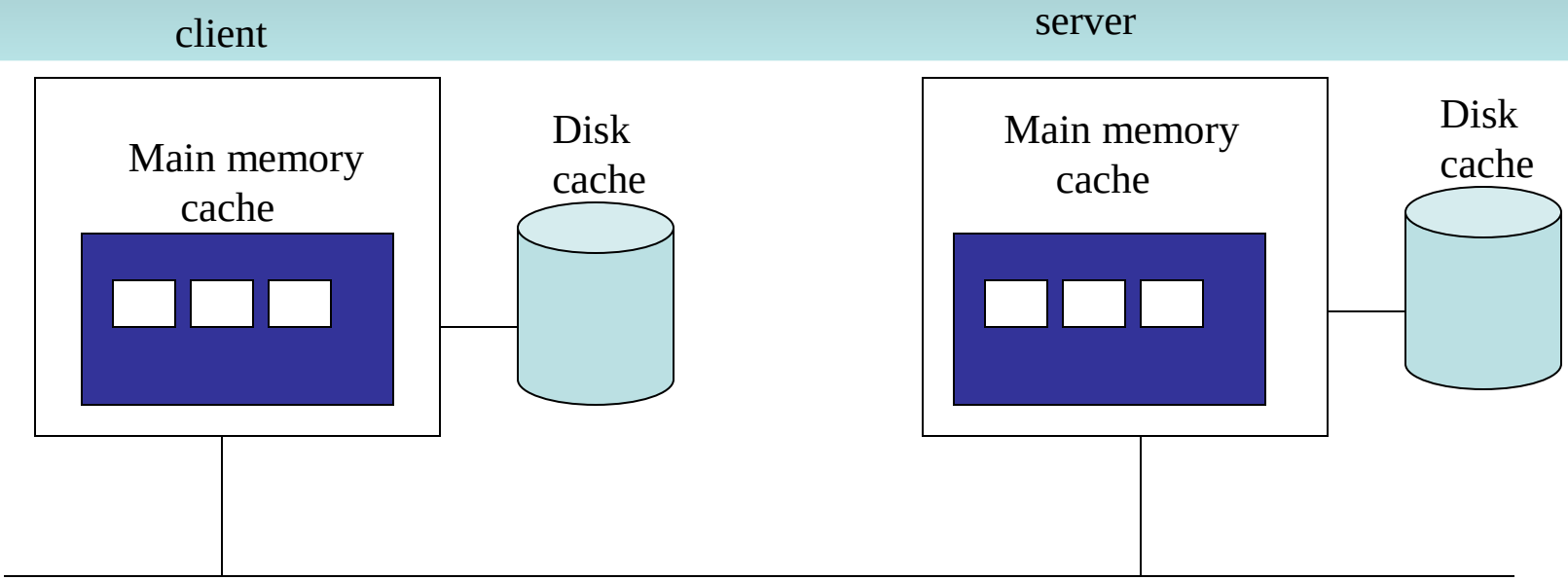
Semantics of File Sharing

- No file update semantics (Immutable files):
  - Files are never updated/modified
  - Allowed file operations: CREATE and READ
  - Files are atomically replaced in the directory
  - Problems:
    - what if two clients want to replace a file at the same time?
      - Take the last one or use any non-deterministic rule
    - Delete file in use by another process

Semantics of File Sharing

- Atomic transactions
  - All file changes are delimited by a Begin and End transaction
  - All files requests within the transaction are carried out in order
  - The complete transaction is either carried out completely or not at all (atomicity)
  - Serializable access
  - Problem: ?
    - Costly to implement

Caching



- Server caching
- Client caching
- Caching unit: files or disk blocks?

Server Caching

- Server Disk
  - + most space
  - + one copy of each file
  - + no consistency problem
  - Issue?
- Server Memory
  - Keep MRU files in server's memory
  - If request satisfied from cache ==> no disk transfer BUT still network transfer
  - Unit of caching?
    - Whole files: pro & con?
      -
    - Blocks: pro & con?
  - What to replace when cache full?
    - LRU

Client Caching

- Disk:
  - Pros & cons?
- Memory:
  - Pros & cons?
- Where to cache
  - user address space
    - Pros & cons?
  - Kernel
    - Pros & cons?
  - separate user-level cache process

Cache Consistency

- Write-through:
  - Every time a block is modified at the client, send modification to server
  - Cache is effective for read traffic but not for write traffic
  - If client cache survives processes, then cache currency has to be validated with the server (version numbers or checksum can be used here)

Cache Consistency

- Delayed write:
  - Make a note that a file has been modified but do not inform the server immediately
  - Send all modifications as a batch to the server every 30 seconds (more efficient)
  - Reduces write traffic for temporary files that are written, read, and deleted before the server needs to be notified
  - Cleaner semantics is traded for better performance

Cache Consistency

- Write on close:
  - Adopt session semantics and write back to the server 30 seconds after file is closed (deleted files are never sent to the server)
  - Still possible for writes to be lost. If two or more processes have the file open for write, only one wins. Similar problem may arise in centralized systems if no locking is issued

Cache Consistency

- Centralized control:
  - A centralized controller keeps track of all files that are open and their respective clients
  - Conflicting requests to open files can be handled in three ways:
    - Deny request
    - Queue request
    - Grant request but notify all clients that have the file open to remove it from their cache and disable caching (unsolicited messages to clients is required)
    - Does not scale and is not robust

Trends in DFS

- Wide area networking
- New hardware
  - Cheap main memory => file system in main memory with backups in videotape or optical disks
  - Extremely fast fiber optic networks => avoid client caching
- Scalability
  - From 100 to 1,000 to 10,000 nodes!
  - Use of broadcast messages should be reduced
  - Resources and algorithms should not be linear in the number of users

Trends in DFS

- Fault tolerance
  - As DSs become more widespread, provisions for higher availability have to be incorporated into the design
- Mobile users
  - Increase in disconnected operation mode
  - Files will be cached for longer periods (hours or days) at the client laptop
- Multimedia
  - New applications such as video-on-demand, audio files pose different demands on the design of a file system

Questions