Distributed File Systems

CS60002: Distributed Systems

Antonio Bruto da Costa
Ph.D. Student, Formal Methods Lab,
Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

What is a Distributed File System

- DFS: distributed implementation of a file system
    - Files, file servers, and users are all dispersed around the network
- Main Goal: DFS should look like a centralized file system to a user
    - Ability to open and update any file on any machine on the network
    - Ability to share files, the same as shared local files

DESIGN GOALS

- Network Transparency
- High Availability
- Scalability
- Concurrency

Design Goals

- Network transparency
  - Same set of file operations for both local and remote files
  - Uniform naming scheme for local and remote files
  - Similar access time for local and remote files
- High availability
  - Files should not become unavailable for “small” failures
- Scalability
  - No. of users, file servers, files handled etc.
- Concurrency
  - Handle concurrent access by different clients in the network

DFS Architecture

- **Client Server Architecture**
  - **Network File System (NFS)**

- **Cluster Based Architecture**
  - **Google File System**

Design Issues : Naming Schemes

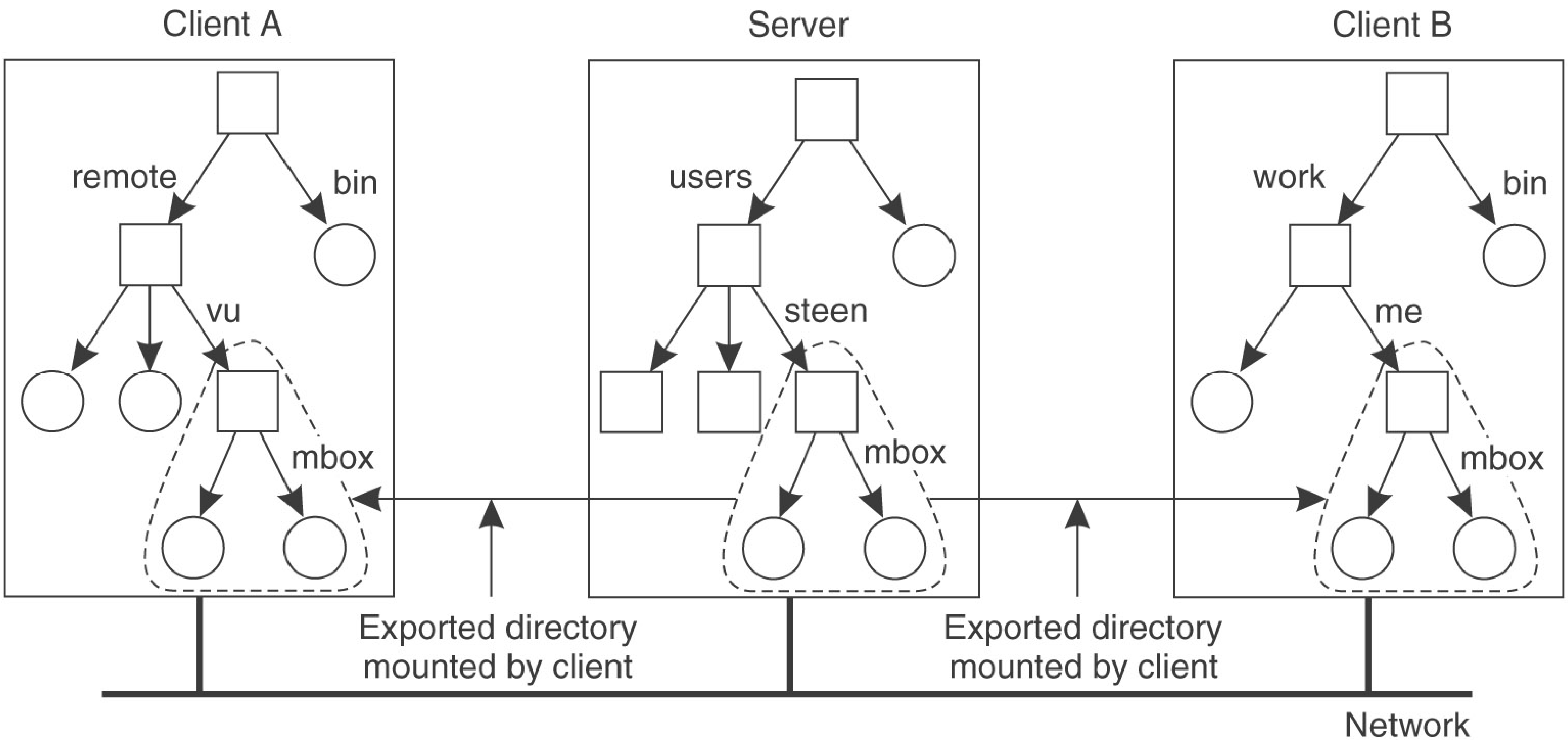
- Files named by combination of host name and local name
  - Not location transparent

- Single global namespace spanning all files
  - Location transparent
  - All systems see the same directory structure
  - A server crash affects all machines
  - Example: Andrew File System (AFS)

DNS Names

- Mount remote directories to local directories
  - Location transparent
  - Directory structure seen by different machines possibly different
  - Mount only when needed, what is needed
  - Mounted remote directories can be accessed transparently
  - Example: Network File System (NFS)

Mounting Remote Directories (An Example: NFS)



Pathname Translation

- Consider /remote/vu/mbox in A
- /remote/vu looked up in A the usual way
- Low level entry for vu contains information that it is actually /users/steen in server
- Request sent to server to look up /users/steen/mbox
- What if the name crosses multiple machine boundaries?

Caching to Improve Performance

- Reduce network traffic by retaining recently accessed disk blocks in local cache
    - Repeated accesses to the same information can be handled locally from the cached copy
- If data accessed not in cache, copy of data brought from the server to the local cache
    - Copies of parts of file may be scattered in different caches.
- Cache-consistency problem – keeping the cached copies consistent with the master file in the presence of write operations

Design Issue: Caching

- Cache location
- Granularity of cached data
- Update policy
- Cache consistency

Cache Location

- In client memory
  - Faster access
  - Good when local usage is transient
  - Enables diskless workstations
- On client disk
  - Good when local usage dominates
  - Can cache larger files
  - Helps protect clients from server crashes

Granularity of Cached Data

- Whole file
  - Easier to maintain consistency
  - High transfer time, specially if only a small part of the file is accessed
- One or more blocks
  - Low transfer time, transfer as and when needed
  - More complex consistency issues
- In general, increasing granularity means higher chance of finding next accessed data, but also higher transfer time

Cache Update Policies

- When is cached data at the client updated in the master file?
- Write-through – write data through to disk when cache is updated
  - Reliable, but poor performance
- Delayed-write – cache and then write to the server later
  - Write operations complete quickly; some data may be overwritten in cache, saving needless network I/O
  - Poor reliability
    - Unwritten data may be lost when client machine crashes
    - Inconsistent data
  - Variation – write dirty blocks back at regular intervals

Cache Consistency

- Is locally cached copy of the data consistent with the master copy?
- Client-initiated approach
  - Client initiates a validity check with server
  - Server verifies local data with the master copy
    - Can use timestamps, version no….
- Server-initiated approach
  - Server records (parts of) files cached in each client
  - When server detects a potential inconsistency, it invalidates the client caches

Sharing Semantics in DFS

- Unix Semantics
  - Read after write
  - Every change becomes instantly visible to all processes
- Session Semantics
  - Same as unix semantics for local clients of a file
  - For remote clients, changes to a file are initially visible to the process that modified the file. Only when the file is closed, the changes are visible to other processes.
- Immutable files

Stateless vs Stateful Service

- Stateless service
  - No open and close calls to access files
  - Each request identifies the file and position in file
  - No client state information in server by making each request self-contained
  - Advantages
    - Better failure recovery
  - Disadvantage
    - Longer request messages, longer processing time

- Stateful Service
  - Client opens a file
  - Server fetches information about file from disk, stores in server memory
    - Returns to client a connection identifier unique to client and open file
    - Identifier used for subsequent accesses until session ends
  - Server must reclaim space used by no longer active clients
  - Advantages
    - Increased performance; fewer disk accesses
  - Disadvantages
    - Poor failure recovery

Network File System (NFS)

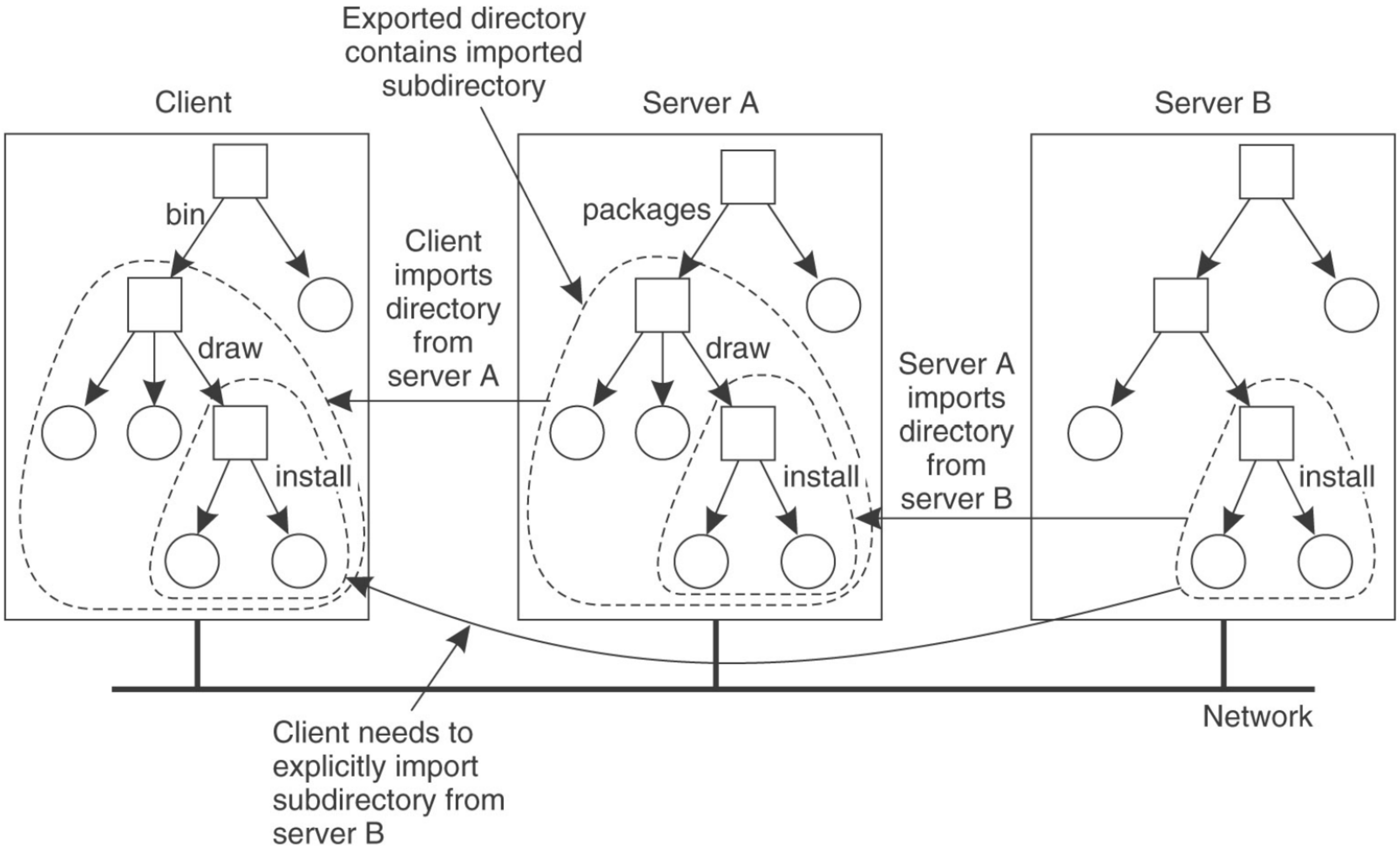
CS60002: Distributed Systems

Antonio Bruto da Costa
Ph.D. Student, Formal Methods Lab,
Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

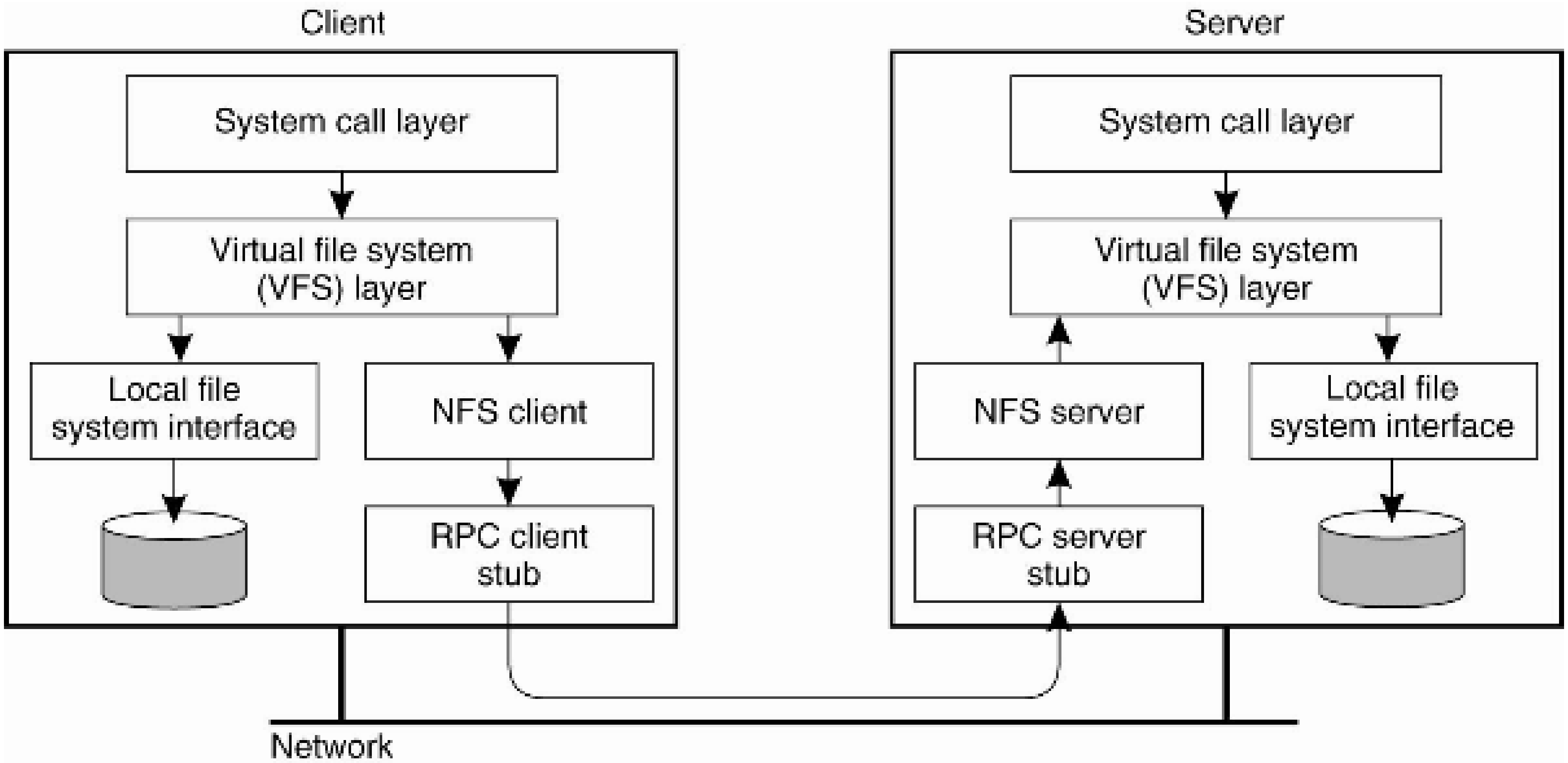
Main Idea

- Server exports parts of a filesystem to clients (Ex. specified in /etc/exports file in Linux)
- Clients "mount" server exported namespace at specific mount points in its namespace tree (specified in /etc/fstab in Linux; can be mounted separately by mount command also)
- Same server filesystem can be mounted at different mount points at different clients, so no single global name space
- Mounts can be cascaded
  - But client must mount from different servers explicitly itself

Nested *Mounting* (NFS v3)



Basic NFS Architecture



Other details

- Location transparent naming (host id needed only during mount)
- Pathname translation
  - Look up each component in the pathname recursively
  - Lookup call made to server when a mount point is crossed
  - Cascading mounts can involve multiple servers during translation
- Set of operations provided for operations on files
  - Lookup, read, write, mkdir, rename, rmdir, readdir, getattr, setattr…
  - No open or close calls (stateless server)
- Most data-modifying calls are synchronous
- Most calls work on an opaque (to client) file handle returned to the client by the server
  - Uniquely identifies the file in the server
- RPC (Remote Procedure Call) used to communicate between server and client

Caching in NFSv3

- Strictly not part of NFS protocol, but used in practice for performance
  - File attributes cached along with file block.
  - Cached file block used subject to consistency checks on file attributes
  - 8 Kb blocks for v2, negotiable for v3
- No locking, no synchronization as part of NFS
  - Separate centralized locking mechanism using lockd
- No clear semantics because of caching nature

NFS v4 Improvements

- Improved access and good performance on the Internet
  - Transit across firewalls, perform well in high latency low bandwidth scenarios, scale to very large numbers of clients per server
- Strong security with negotiation built into the protocol.
  - Supports the RPCSEC_GSS protocol, provides a mechanism for clients and servers to negotiate security parameters, requires clients and servers to support a minimal set of security schemes
- Good cross-platform interoperability

NFS v4: COMPOUND Operation

- Allows building more complex requests by combining one or more traditional NFS procedures
- Example: read data from a file in one request by combining LOOKUP, OPEN, and READ operations in a single COMPOUND RPC.
- Reduces the number of RPCs needed for logical file system operations
- The operations combined within a COMPOUND request are evaluated in order by the server. Once an operation returns a failing result, the evaluation ends and the results of all evaluated operations are returned to the client
- Still uses filehandle as the id of the file. Passes a filehandle from one operation to another within the sequence of operations

NFS v4: Open and Close

- The NFS version 4 protocol introduces OPEN and CLOSE operations
  - Stateful model
- The OPEN operation provides a single point where file lookup, creation, and share semantics can be combined
- The CLOSE operation also provides for the release of state accumulated by OPEN.

NFS v4: Locking

- Locking
  - No separate lock manager as in v3
  - Lease based locking model as part of NFS
  - State associated with file locks maintained at the server

NFS v4: Caching

- The client checks validity of cached file data when the file is opened
  - Checks with server if the file has been changed
  - Based on this, the client determines if the cached file data should kept or released
- When the file is closed, any modified data is written to the server
- Serializing access to file data can be done through locking

NFS v4: Open Delegation

- Server can delegate some responsibilities to the client for a file
- At OPEN, the server may provide the client either a read or write delegation for the file
  - Assures certain semantics to the client
- If the client is granted an open delegation, the client machine can locally handle open and close operations from other clients in the same machine.
- Allows the client to locally service operations such as OPEN, CLOSE, LOCK, LOCKU, READ, WRITE without immediate interaction with the server

NFS v4: Recalling Delegation

- Delegations can be recalled by the server
- If another client requests access to the file such that the access conflicts with the granted delegation, the server can recall the delegation from initial client
- Problems with delegation and stateful servers
  - Failure handling

NFS v4: Replication

- locations attribute at server can store the location of a file system
- If a file system is migrated in between, the client will receive an error when operating on the file system
- The client can then probe the server query for the new file system location.
- The attribute can also hold multiple server locations to allow replication of file systems
  - Client can use its own policies to access one of the replicas

NFS v4: Security

- Use of RPCSEC_GSS
  - Security framework to support multiple ways of setting up secure channels
  - Supports message integrity and confidentiality
- Should be configured with Kerberos
  - Support for authentication
- It must also support a method LIPKEY
  - LIPKEY is a public key system
  - Clients are authenticated using passwords
  - Servers are authenticated using a public key