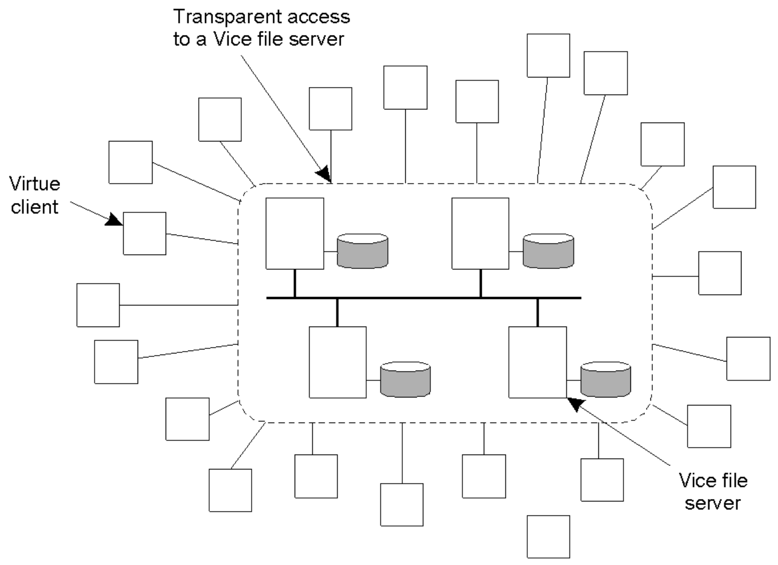
Today: Coda, xFS

- Case Study: Coda File System
- Brief overview of other recent file systems
  - xFS
  - Log structured file systems

Coda

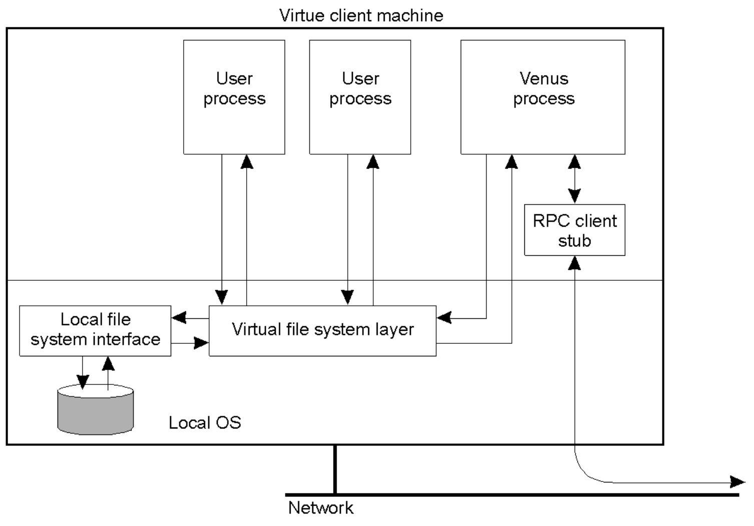
- Coda: descendent of the Andrew file system at CMU
  - Andrew designed to serve a large (global community)
- Salient features:
  - Support for disconnected operations
    - Desirable for mobile users
  - Support for a large number of users

Overview of Coda



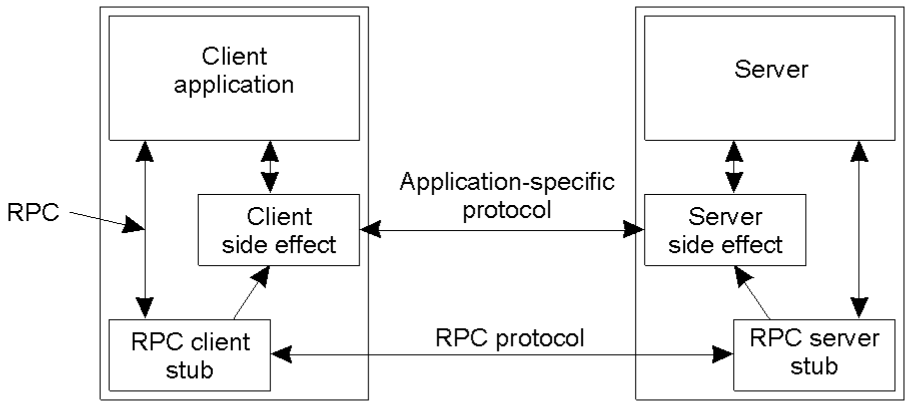
- Centrally administered Vice file servers
- Large number of virtue clients

Virtue: Coda Clients



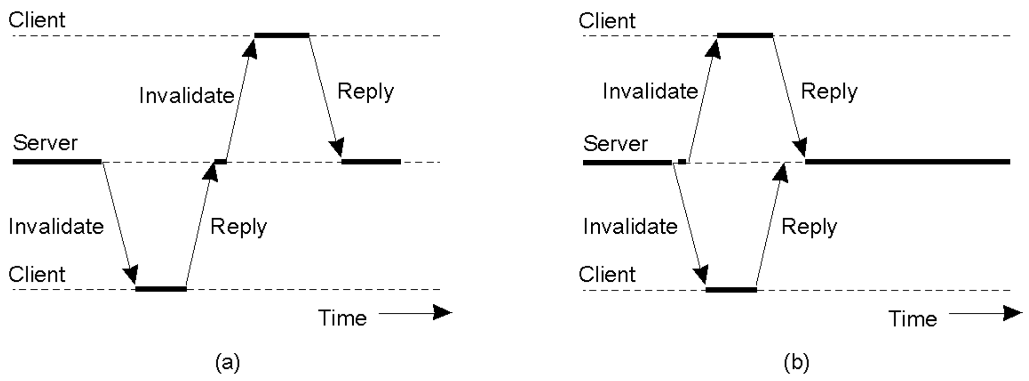
- The internal organization of a Virtue workstation.
  - Designed to allow access to files even if server is unavailable
  - Uses VFS and appears like a traditional Unix file system

Communication in Coda



- Coda uses RPC2: a sophisticated *reliable* RPC system
  - Start a new thread for each request, server periodically informs client it is still working on the request
- RPC2 supports *side-effects:* application-specific protocols
  - Useful for video streaming [where RPCs are less useful]
- RPC2 also has multicast support

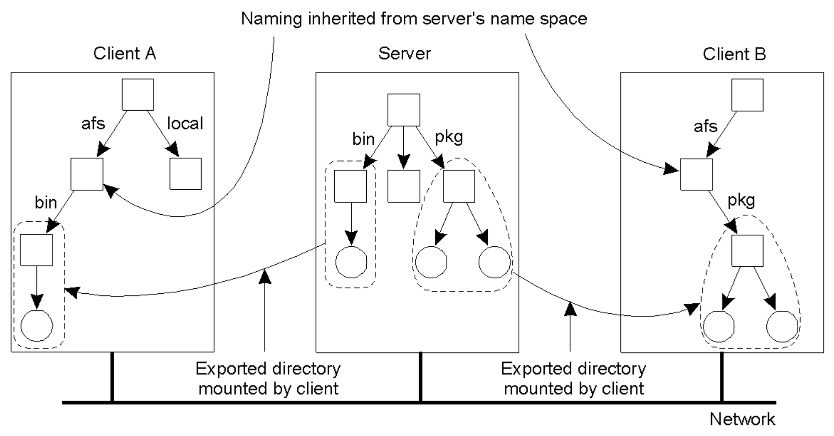
Communication: Invalidations



a) Sending an invalidation message one at a time.
b) Sending invalidation messages in parallel.

Can use MultiRPCs [Parallel RPCs] or use Multicast
- Fully transparent to the caller and callee [looks like normal RPC]

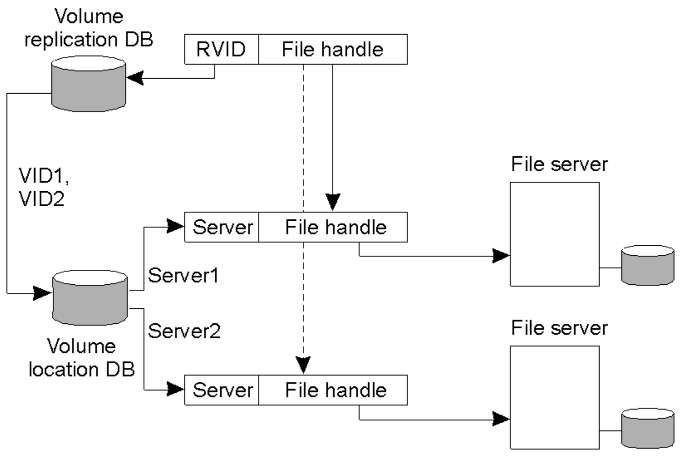
Naming



Naming inherited from server's name space
Client A
Server
Client B
afs
local
bin
bin
pkg
afs
pkg
Exported directory mounted by client
Exported directory mounted by client
Network

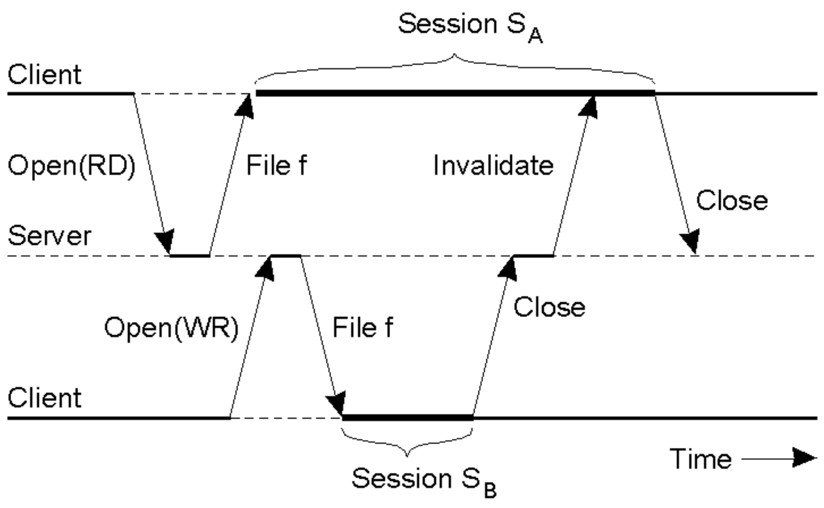
- Clients in Coda have access to a single shared name space
- Files are grouped into *volumes* [partial subtree in the directory structure]
  - Volume is the basic unit of mounting
  - Namespace: /afs/filesrv.cs.umass.edu [same namespace on all client; different from NFS]
  - Name lookup can cross mount points: support for detecting crossing and automounts

File Identifiers



- Each file in Coda belongs to exactly one volume
  - Volume may be replicated across several servers
  - Multiple logical (replicated) volumes map to the same physical volume
  - 96 bit file identifier = 32 bit RVID + 64 bit file handle

Sharing Files in Coda



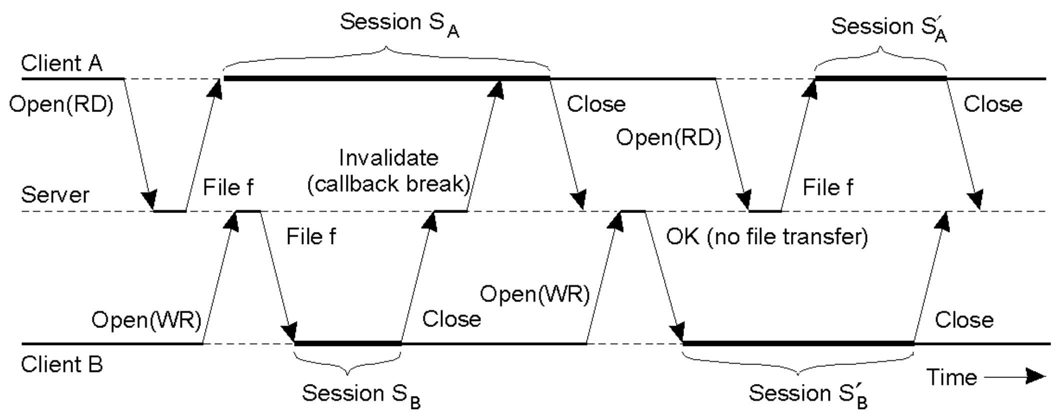
- Transactional behavior for sharing files: similar to share reservations in NFS
  - File open: transfer entire file to client machine [similar to delegation]
  - Uses session semantics: each session is like a transaction
    - Updates are sent back to the server only when the file is closed

Transactional Semantics

File-associated data	Read?	Modified?
File identifier	Yes	No
Access rights	Yes	No
Last modification time	Yes	Yes
File length	Yes	Yes
File contents	Yes	Yes

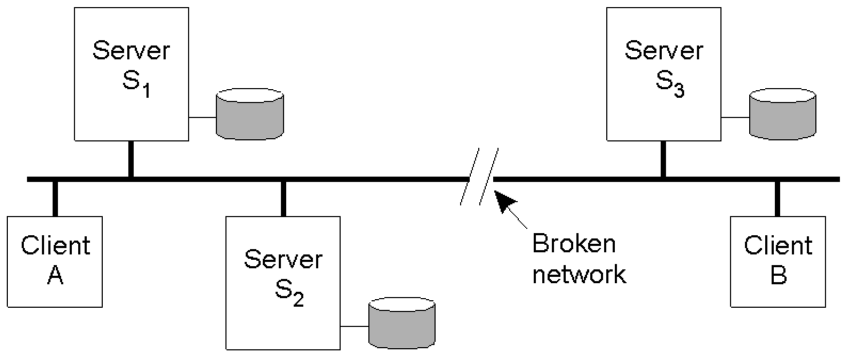
- Network partition: part of network isolated from rest
  - Allow conflicting operations on replicas across file partitions
  - Reconcile upon reconnection
  - Transactional semantics => operations must be serializable
    - Ensure that operations were serializable *after thay have executed*
  - Conflict => force manual reconciliation

Client Caching



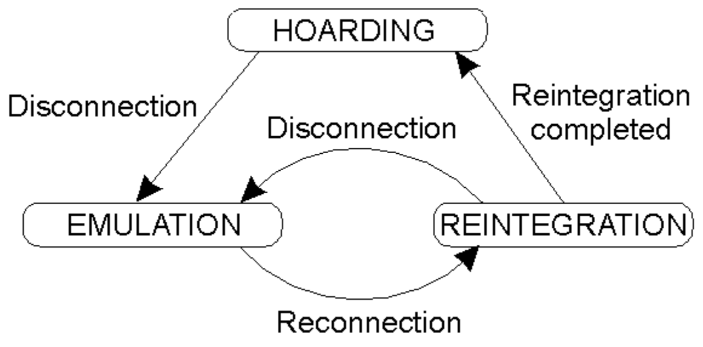
- Cache consistency maintained using callbacks
  - Server tracks all clients that have a copy of the file [provide *callback promise*]
  - Upon modification: send invalidate to clients

Server Replication



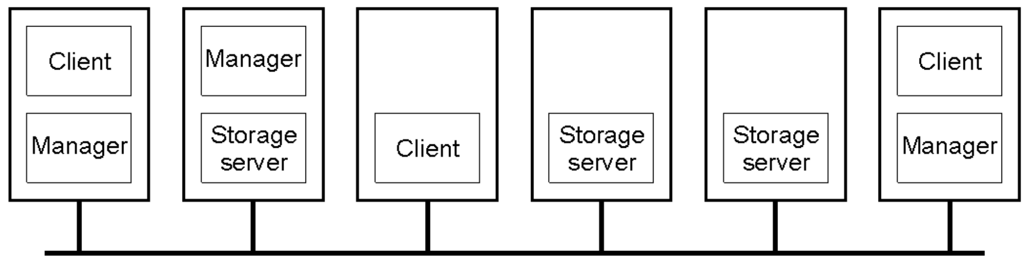
- Use replicated writes: read-once write-all
  - Writes are sent to all AVSG (all accessible replicas)
- How to handle network partitions?
  - Use optimistic strategy for replication
  - Detect conflicts using a Coda version vector
  - Example: [2,2,1] and [1,1,2] is a conflict => manual reconciliation

Disconnected Operation



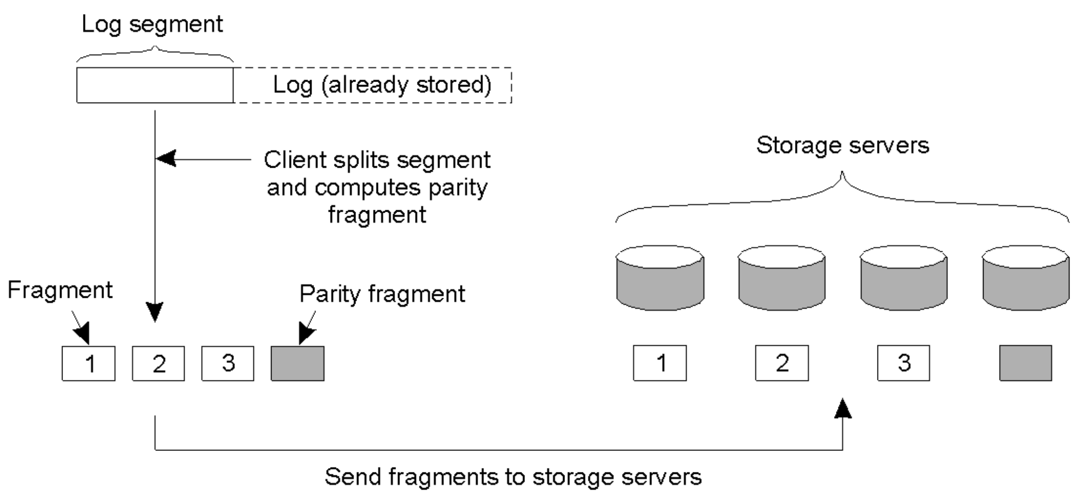
- The state-transition diagram of a Coda client with respect to a volume.
- Use hoarding to provide file access during disconnection
  - Prefetch all files that may be accessed and cache (hoard) locally
  - If AVSG=0, go to emulation mode and reintegrate upon reconnection

Overview of xFS.



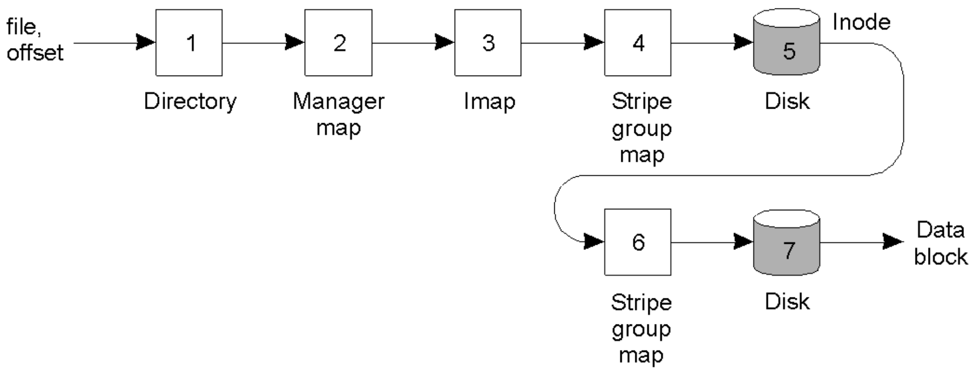
- Key Idea: fully distributed file system [*serverless* file system]
- xFS: x in “xFS” => no server
- Designed for high-speed LAN environments

Processes in xFS


Log segment
Log (already stored)
Client splits segment and computes parity fragment
Storage servers
Fragment
Parity fragment
1
2
3
1
2
3
Send fragments to storage servers

- The principle of log-based striping in xFS
  - Combines striping and logging

Reading a File Block



- Reading a block of data in xFS.

xFS Naming

Data structure	Description
Manager map	Maps file ID to manager
Imap	Maps file ID to log address of file's inode
Inode	Maps block number (i.e., offset) to log address of block
File identifier	Reference used to index into manager map
File directory	Maps a file name to a file identifier
Log addresses	Triplet of stripe group, ID, segment ID, and segment offset
Stripe group map	Maps stripe group ID to list of storage servers

- Main data structures used in xFS.