CS425 / CSE424 / ECE428 — Distributed Systems — Fall 2011

Decentralized File Systems

Some material derived from slides by Prashant Shenoy (Umass) & courses.washington.edu/css434/students/Coda.ppt

Outline

CODA

Distributed revision control

AFS Review

- Assumptions
  - Clients have disks
  - Read/write & write/write conflicts are rare
- Techniques
  - Whole-file long-term caching
  - Leases / promises to operate w/o server contact for 15 minutes

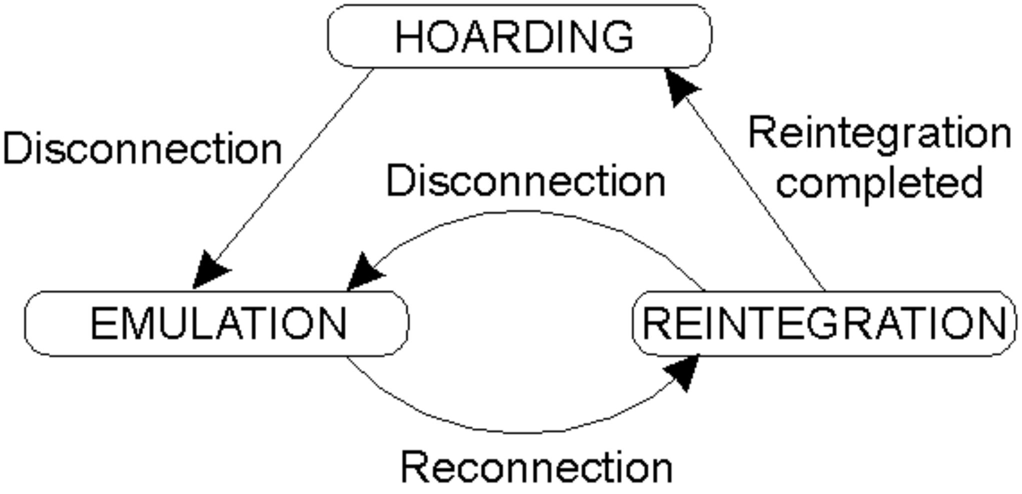
CODA idea: extend to longer than 15 minutes

CODA Approach

- Many replicas
  - Servers: 1st class replicas
    - Unlike AFS, more than one, even with read/write
  - Clients: 2nd class replicas
- Each **volume** has a *volume server group (VSG)*
- *Available VSG (AVSG):* reachable members of VSG
  - AVSG = VSG (Normal operation)
  - AVSG $\subsetneq$ VSG (Partition)
  - AVSG = $\varnothing$ (Disconnected operation)

CODA clients

- Three states:
  - Hoarding: cache files aggressively
  - Emulation: operate in disconnected mode, satisfy read/write requests from cache
  - Reintegration: propagate local changes back to servers



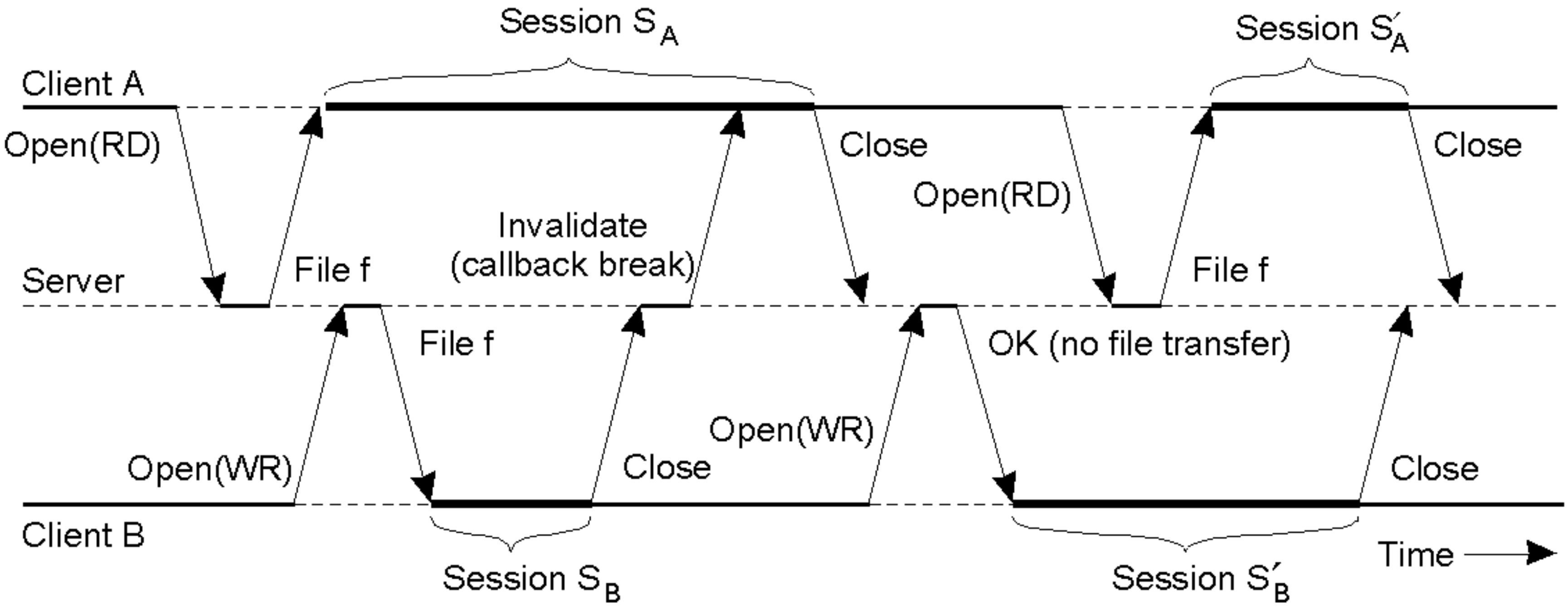
Hoarding

- Occurs during normal, connected operation
- Add to cache:
  - Files that are accessed
  - Files in Hoard Database (HDB) – user specified
- Maintain leases (promises) on cached files
- On lease break:
  - Immediately fetch new file?
  - Wait until next reference?

Hoard Walk

- Periodically (every 10 minutes)
  - Walk cache & hoard database
  - Refresh any invalidated files
    - If not refreshed on demand
  - Restore equilibrium in cache
- Priorities
  - HDB specifies hard-coded priorities
  - Recently access files obtain (decaying) priority
  - Equilibrium: pri(file in cache) > pri(file not in cache)

Client Caching



- Callback break **only** after close

Disconnected Operation

- Local cache *emulates* server
  - Serves files from cache
    - Cache miss = error
  - Performs access checks
  - Stores writes in replay log
- Replay log
  - Stored in Recoverable Virtual Memory (stable storage)
  - History of directory operations
  - Last write to a file (remember, whole file update semantics)

Reintegration

- Replay changes stored in log
  - Merge directory operations
  - Execute (last) file storage operation
- Update cached files based on server version
- Conflicts (write/write only)
  - Abort reintegration
  - Send log for user for manual resolution
- “Future thoughts”
  - Automatic conflict resolvers
  - Unit of integration < volume

Opens

- A successful open means:
  - The file received is the latest version.
  - Or there was 1+ lost callbacks and the file received is the latest version within the t seconds of a Venus server probe.
  - Or the client is disconnected but the file is cached
- A failed open means:
  - There is a conflict that must be manually resolved
  - Or the client is disconnected and the file is not cached

Closes

- A successful Close means:
  - All members of the AVSG have received the latest version of the file.
  - Or the client is disconnected.
- A failed Close means:
  - There is a conflict in the AVSG that must be manually resolved
    - Because the file originally received was not current
    - Or because the AVSG expanded and gained a modifed version of the file
  - A Close will always succeed if the client is disconnected

Replica Consistency

- Consistency strategy:
  - Read one/write all
  - Available copies replication
- For reads: preferred server (based on latency, load, etc.)
- For writes: all servers in AVSG

Partitions

- Each file has a Coda Version Vector (CVV)
- Incremented by each server at each update
  - E.g., initial value: [1,1,1]
  - Write to servers 1,2: [2,2,1]
  - Write to server 3 [1,1,2]
- At reconnection:
  - [1,1,2] and [2,2,1] => conflict
  - Manual resolution!

CODA Redux

- Enable disconnected operation & handle partitions
- Use *optimistic* cache consistency
- *Manual* conflict resolution
    - Assumption (validated): write/write conflicts are rare
- What if they aren't?

Version Control Systems

- Used for managing large software projects
- Properties:
  - Many developers: frequent write/write conflicts
  - Changes both fix & introduce bugs
    - Useful to "unroll" changes
    - Useful to keep history of files

RCS

- Revision Control System (RCS)
- Pessimistic sharing workflow
  - co file [locks copy]
  - [edit file]
  - ci file [commits changes, unlocks copy]
- Unit of control: single file
- Storage: single filesystem

CVS

- Based on RCS, but more “advanced”
- Unit of control: directory
- Client-server architecture
- Optimistic sharing workflow:
  - Checkout [no lock, done once]
  - Update [receive latest version]
  - Edit
  - Commit
- If conflict
  - Edit
  - Commit => conflict
  - Update – *merge* changes
  - Commit

Merging

- Mostly automated
  - Maintain diffs / patches between versions
  - Record context of edits
  - Replay edits if context can be identifies
- Conflicts still exist
  - But more rare
  - To be resolved manually

Distributed Version Control System

- Every copy is a full repository
  - Peer-to-peer architecture
- Revisions committed to local copy
  - “Replay log” maintained locally
- Bi-directional exchange of changes
  - Between any two repositories
  - Complex workflow possible

Example: Git workflow

- Obtain local copy:
  - git clone repository-url
- Make local edits
  - edit
  - git commit
  - edit
  - git commit
- Update local copy
  - git pull repository
  - Merge any remote and local changes
- Update remote copy
  - git push repository

Other concepts

- Branching
- Rebasing
- Tags
- …