CS 525M – Mobile and Ubiquitous Computing Seminar

Bradley Momberger

The Evolution of Coda

- Attributions
  - Title: “The Evolution of Coda,” 2002.
  - Author: M. Satyanarayanan, Carnegie Mellon University
  - Published in “ACM Transactions on Computer Systems,” May 2002.

What is Coda?

- Coda is a robust networked file system
- Coda has several features that ensure high availability
  - Redundancy across several servers
  - Local caching for clients
  - Disconnected and weakly connected operation
  - Transactions and conflict resolution

- Coda has additional features inherited from prior filesystem projects
  - Location-independent namespace
  - Sophisticated access control
  - Implementation as a user-space runtime

Prologue: AFS

- Andrew File System, or AFS, is the precursor to Coda.
- Key Features
  - Client/server model
  - Universal namespace for files: /afs/*site*/*top_level*/...
  - Directory-level access control lists: “rlidwka” format
  - Client-side caching of full files
  - User-level implementation, with Venus caching client.
- AFS shares these core elements with Coda, but lacks Coda's redundancy across servers or clients.
- Worst-case scenario: In a networked environment, a user's home directory across all servers is stored on AFS, and the containing volume becomes temporarily unavailable.
- The availability problem of AFS was the original impetus for developing Coda

Coda: Server Replication

- Two schools of thought for server replication
  - Pessimistic Replication: Ensure consistency of files across all servers before allowing updates.
    - Guaranteed to behave as a local, single-user file system for updates
    - Will lock files as long as file replicas are inconsistent.
  - Optimistic Replication: (Used by Coda) Provide client availability despite inconsistency concerns.
    - Resolve conflicts after inconsistent updates.
    - Transaction logs as the primary vehicle of resolution
- Coda uses optimistic replication; the designers were primarily concerned with high availability of files.

Server Replication: Key Concepts

- Optimistic replication on read:
  - Client grabs copy from one volume server, but checks version status of file on all volume servers.
  - Resolve conflict at server level before completing read request.
- Optimistic replication on write:
  - Client moves status and file data to each server in parallel, and receives a datagram in return. (COP1)
  - Client then returns to all servers the list of servers that returned the datagram. (COP2)
  - Discrepancy between versioning can then be resolved between the servers based on transaction history

Server Replication: Key Concepts

- Optimizations
  - COP1 (update) phase done in sequence, but COP2 (verification) is done asynchronously after control is handed back to user.
  - RPCs to Coda servers are handed out in parallel fashion; all COP1 requests are done simultaneously.
- Transaction control
  - When servers in the Volume Server Group (VSG) are disconnected, they use a transaction control method to log updates to server data structures.
  - Transaction control is implemented in a lightweight library developed in-house, called Recoverable Virtual Memory (RVM).

Coda: Disconnected Operation

- Disconnected Operation
  - Server replication is not useful when the client is disconnected from the network completely
    - Also in the case that all servers in the VSG fail.
  - The need for a client to work without server access is essential to Coda's applicability to mobile computing.
    - A mobile device will often be out of service range, or have little to no connection to the VSG.
    - Mobile devices conserve energy though limiting network connections.
- In Coda, no distinction is made between involuntary and voluntary client disconnections.
  - This aids in Coda's applicability to mobile computing.

Disconnected Operation: Stages

- Three stages of disconnected operation
  - Hoarding (connected state)
    - Coda caches files locally using a combination of a standard LRU policy and user assistance through a Hoarding Database (HDB).
  - Emulating (disconnected state)
    - Venus acts as the Coda file server.
    - Cache misses on read are treated as filesystem failures
    - Updates are logged in the Client Modification Log (CML)
  - Reintegrating
    - Per-volume repopulation of file data on Coda servers.

Coda: Conflict Resolution

- Conflict Resolution
  - Because of disconnected operation, files in a Coda tree cannot be assumed to have consistency across all clients and servers.
  - The chance of a file having a version fork is small but nonzero. (~0.25% assuming daily cache update)
  - In the case of directory structure conflicts, Coda servers handle conflict resolution without user intervention.
    - Use CML when resolving disputes after disconnected operation
    - Use RVM when resolving disputes across servers.
    - One server in the VSG appointed as the resolution coordinator merges the available logs and sends the merged log back to the other servers.

Conflict Resolution: Files

- Conflict resolution (continued)
- In the case of file conflicts, two possible methods exist for resolving conflicts.
  - A Coda-aware application can use the application-specific resolver (ASR) framework to resolve its own file conflicts, based on file data, without user intervention.
  - If the ASR fails or there is no ASR, Coda alerts the user by representing the conflicting file as a dangling symlink. When the user initiates the recovery process, he is able to see each server's copy of the file and choose the correct one.

Coda: Weak Connection

- Weakly Connected Operation
  - Weak connection, for example remotely via modem or over a cellular network, is beneficial to minimizing version conflicts and data loss.
  - Ideally, provisions for weak connection would not be necessary, but the limits of mobile devices force the issue.
  - Performance degradation, especially for strongly connected clients, is a major issue in receiving updates via weak (i.e. slow, impermanent) connections.
    - Synchronization with updates from weak connections would cause long unavailability periods for strong connections.

Weak Connection: Assistants

- Weak updates in Coda have two helper features:
  - Volume level hashing for rapid validation
    - Before doing any validation on the individual files in a volume, do a validation of the entire volume to see if you need to validate files.
  - Trickle Reintegration
    - Replace the “Reintegrating” phase mentioned before with a “Write Disconnected” phase.
    - During write-disconnected phase, set a time to allow for changes listed in the CML to cancel themselves out. Initiate updates only after the time has elapsed.
    - Break data and transaction uploads into chunks, sized as large as the connection can feasibly upload in 30 seconds.

Coda: Translucent Cache

- Translucent (as opposed to transparent) caching
  - Use of simple model and control panel to provide user with notification of cache issues and connection state
  - Control over cache management
    - User patience model: predict that based on time needed to update cache over weak connection, the user will not want to allow automatic cache management.
    - Cache notification GUI: provide the user with a moderate amount of detail about the state of the Coda cache (full, updated, conflict, etc).

Coda: Future Evolution

- Future evolution
  - Coda is the filesystem backing Aura, the context-aware computing project ongoing at CMU.
  - Contributions to the Open Source movement.
    - Coda source and runtimes freely available
    - 1999 *Linuxworld* Editor's award for file management.
  - Ongoing projects
    - Primary focus of current Coda development is wide availability and distribution (http://coda.cs.cmu.edu)
    - Ports to other platforms, including Windows
    - Using surrogates, like strongly connected user stations, for proxy cache.