Consistency of Replicated Data in Weakly Connected Systems

CS444N, Spring 2002

Instructor: Mary Baker

How will people use mobile computers?

- Traditional client of a file system?
  - Coda, Ficus
- Client of generalized server?
  - Bayou
- Xterm?
- Stand-alone host on the Internet?
  - Mobile IP, TRIAD
- Divisions not clear-cut

Evolution of wireless networks

- Early days: disconnected computing (Coda'91)
  - Laptops plugged in at home or office
  - No wireless network
- Now: weakly connected computing (Coda, Bayou)
  - Assume a wireless network available, but
  - Performance may be poor
  - Cost may be high
  - Energy consumption too high
  - Intermittent disconnectivity causes involuntary breaks
- Future: (Some local research)
  - Breaks will be voluntary?
  - Exploit weak connectivity further

Data replication

- Replication
  - Availability: network partition
  - Performance: go to closest replica
- Caching
  - Performance
  - Coda: for availability too in disconnected environment
- Difference between caching and replication?
  - Replica is considered a primary copy
  - Division not always sharp

Use of disconnected computing

- Where does it work?
  - Wherever some information is better than none
  - Where availability more important than consistency
- Where does it not work?
  - Where current data is important
- Traditional trade-off between availability and consistency
  - Grapevine
  - Sprite
- Consistency has also been traded for other reasons
  - NFS (simplicity, crash recovery)

Retrofitting disconnection

- Disconnection used to be rare
  - Much software assumes it is a rare error condition
  - Okay for system to stall
- Locus and other systems used a lot of consensus algorithms among replicas
  - Replicas may not be reachable
  - Latency of chatty protocols not acceptable
- Perfect consistency no longer always reasonable
  - Sprite
- Michigan Little Work project: no system mods
  - Integration must be based on individual files
  - Integration not transactional

Coda assumptions

- Blend between individual robustness and infrastructure
- Clients are appliances
  - Vulnerable, unreliable, security problems, etc.
  - Don’t treat as primary location of data
  - Assume central computing infrastructure
- Client self-sufficient
  - Hoarding
  - Allow weak consistency
  - Off-load servers with work on clients
  - Time-limited self-sufficiency

In practice

- Does this work?
  - Lots of folks keep main copy on laptops
  - Which address book is primary copy?
  - Multiple home bases for computing infrastructure
- Bayou treats portables as first-class servers
  - Replication for caching purposes as well
- Some centralization would be useful
  - Personal metadata?

Hoarding

- Coda claims users are good at predicting their needs
  - Already do it for extended periods of time
  - Can help with automated hoarding
- Cache miss on /var/spool/xxx33.foo
  - What do you do?
- Information for hoarding included in RPM packages?

Conflict resolution

- Coda:
  - Transparent where possible
  - Okay to ask user
- Bayou:
  - Programmatic conflict resolution
  - May in fact ask user
- How do we incorporate user feedback?
  - Early? At conflict time?
  - File-type specific information?
  - Transparent at what level? User? Appl? OS?
  - What can a user really do?

Replica control strategies

- Optimistic: allow reads and writes and deal with damage later
  - Good availability
- Pessimistic: don't allow multiple access so no damage can occur
  - Availability suffers
- All depends on length of disconnections and whether they are voluntary or not
- One client out with lock for a long time not okay
- Bayou avoids this

Other topics

- Call-back breaks
  - During disconnection
- Log optimization
- User patience threshold
- Per volume replay log
  - Inter-volume dependencies?
- Conflict measurements
  - Same user doesn't mean no conflict!
  - 0.25% still pretty high!

Write-sharing

- Types of write-sharing: sequential, concurrent
- Sequential
  - User A edits file
  - User B reads or edits file
  - Updates from A need to get to B so B sees most recent data
  - NFS: Window of time between two events determines consistency, even with “almost write-through” caching
  - Sprite/Echo/etc.: Second event may generate a call-back for data write-back and/or token

Write-sharing, continued

- Concurrent:
  - Two hosts edit or read/edit the same file at the same time
  - Sprite turned off caching to maintain consistency
- What does “the same time” really mean?
  - Open/close?
  - Duration of lease?
  - Explicit lock?
  - Echo read/write tokens make all sharing sequential

How much sharing?

- Sprite:
  - Open/close mechanism with callbacks
  - 0.34% of file opens resulted in concurrent write-sharing
  - 1.7% of file opens result in server recall of dirty data (concurrent or sequential)
- Would weaker (NFS) consistency work?
  - With 60-second window, 0.34% of opens result in potential use of stale cache data with 63% of users affected
- AFS:
  - "Only" 0.34% of sequential mutations involve 2 users
  - (But one user can cause conflicts with himself!)

Replica control strategies

- Optimistic: allow reads and writes
  - Deal with damage later
  - Good availability
- Pessimistic: don't allow multiple access
  - No damage can occur
  - Availability suffers
- Choice depends on
  - Length of disconnections
  - Whether they are voluntary
  - Workload and applications
- One client off with lock for a long time not okay

Coda callbacks: optimistic

- Client A caches copy, registers callback
- Client B accesses file: server performs callback break to A
- When connected: client discards cached copy
- Intended for strongly connected world
- When disconnected, client doesn't see call-back break
- Must revalidate files/volumes on reconnection
- This is where room for conflicts arises
- Even when weakly connected, client ignores call-back break!

Callback breaks, continued

- On hoard walk, attempt to regain callbacks
  - Instead of regaining them earlier
- Modified files likely to be modified again
  - Avoid traffic of many callbacks
- Volume callbacks helpful at low bandwidth

Log optimization in Coda

- Per-volume replay log
- Optimizations: rmdir cancels previous mkdir and itself
- Overwrites of files cancel previous file writes
- Why such a range in compressibility?
  - Some traces only 20%
  - Others 40-100%
  - Hot files?
- Inter-volume dependencies?

Impact of trickle reintegration

- Too large a chunk size interferes with other traffic
  - Partly a result of whole-file caching
  - Whole-file caching good for avoiding misses
  - Better refinement for reintegration?
- How useful is think time notion in trace replay results?
  - Why not just measure a few traces and correlate those to reality?
- Other possible optimizations?
  - File compression?
  - Deltas?

Cache misses in Coda

- If disconnected, either return error to program or stall
- Modeling user patience threshold
  - Goal: improve usability by reducing frequency of interaction
  - When confident of user's response, don't contact user
  - Willing to wait longer for more important file
  - Why isn't this sensitive to overall amount of waiting? (Other misses too)

Other design choices?

- Coda: existence of weakly connected clients should not impact other clients
- Instead: examine choice of some amount of impact
- Exploit weak connectivity for better consistency?
- Use modified form of Leases?
  - Attempt to reintegrate modifications
  - Use leases to help clients determine which files to reintegrate
- Maybe choose to stall new clients for length of reasonable lease

Numbers in Coda paper

- Nice attempt to model tricky things
- Hard to see how we can use these actual numbers outside this paper
- Transport protocol performance comparison looks iffy
  - Maybe due to measurements on Mach

Bayou session guarantees

- Lack of guarantees in ordering reads/writes can confuse users and applications
- A user/application should see sensible world during period of a “session”
- How we implement/define sessions is interesting part

Bayou environment

- Bayou: a swamp of mobile DB “servers” moving in and out of contact with each other
- Pair-wise contact between any of them
- Read-any/write-any base
- Eventual consistency relies on
  - Total propagation: Assumes “anti-entropy” process: there exists some time at which a write is received by all servers
  - Consistent ordering: all servers apply non-commutative writes to their databases in the same order

Bayou environment, cont.

- Operation over low-bandwidth networks
- Only updates unknown to receiver propagate
- Incremental progress
- One-way direction of updates
- Efficient storage (can discard logged updates)
- Propagation through transportable media
- Light-weight management of dynamic replica sets
- Propagate operations, not data

Anti-entropy assumptions

- Each new write from client to a server gets "accept stamp" including:
  - Server ID of accepting server
  - Time of acceptance by that server
- Each server maintains version vector V about its update status
  - Server S's V[serverID] contains largest write known to S received from a client by serverID
- Assume all servers keep log of all writes received
  - They don't actually keep all writes forever
- Prefix property:
  - If S has write w accepted from some client by X
  - Then S has all writes accepted by X prior to w

Anti-entropy algorithm

Algorithm for S to update R

S gets R's version vector

For each write w in S's write log {

For the server that stamped w, does R have all the writes up to and including w?

If not, update R

}

Write-log management

- Can discard “stable” or “committed” writes
  - Writes whose position in log will not change
- Trade-off between storage and bandwidth
  - May have to send whole DB to client gone a long time
- Bayou uses a primary replica to commit writes
  - Commit sequence number provides total ordering on writes
- Prefix property maintained
  - Uncommitted writes treated as before
  - Committed writes propagated before tentative ones
- Write-log rollback required
  - On sender if sender has to send whole DB to receiver
  - On receiver to earliest write it must receive

Guarantees for sessions

- Read your writes
- Monotonic reads
- Writes follow reads
- Monotonic writes

Read your writes

- A session's updates shouldn't disappear within that session
- Example errors:
  - Missing password update in Grapevine
  - Reappearing deleted email messages

Monotonic reads

- Disallow reads to a DB less current than previous read
- Example error:
  - Get list of email messages
  - When attempting to read one, get “message doesn’t exist” error

Writes follow reads

- Affects users outside session
- Traditional write/read dependencies preserved at all servers
- Two guarantees: ordering and propagation
  - Order: If a read precedes a write in a session, and that read depends on a previous non-session write, then previous write will never be seen after second write at any server. It may not be seen at all.
  - Propagation: Previous write will actually have propagated to any DB to which second write is applied.

Writes follow reads, continued

- Ordering - example error:
  - Modification made to bibliographic entry, but at some other server original incorrect entry gets applied after fixed entry
- Propagation - example error:
  - Newsgroup displays responses to articles before original article has propagated there

Monotonic writes

- Writes must follow any previous writes that occurred within their session
- Example error:
  - Update to library made
  - Update to application using library made
  - Don’t want application depending on new library to show up where new library doesn’t show up

SyncML

- Pair-wise contact between any source/sink of data
- No support for eventual consistency between all replicas
- Takes into account network delay and BW
  - Ideally one request/response exchange
  - Request asks for updates and/or sends updates
  - Response includes updates along with identified conflicts and what to do about them
- Handles disconnection during synchronization

Some parameters of synch schemes

- What is a client/server?
- Who can talk to whom?
- Support for multiple replicas?
- Transparent
  - Replication?
  - Synchronization?
  - Conflict management?
- Consistency constraints
  - Time limits or eventual consistency?
  - All replicas eventually consistent?

Parameters, continued

- Whole file?
- Vulnerabilities
  - Crash during sync?
  - Bad sender/receiver behavior?
  - Authentication isn't enough to predict behavior