

Mobile File Systems (Disconnected Operation)

Anthony D. Joseph
CS 294-1

Lecture 13
November 22, 2000

Outline

- Scribe?
- Project poster session: Dec 11 2-4pm
- Project papers due Dec 12 5pm (10-12 pgs)
- Fundamental Challenges in Mobile Computing [Satyanarayan]
- A Highly Available File System for a Distributed Workstation Environment [Satyanarayan]
- DBMate [Badrinath]

Fundamental Challenges in Mobile Computing [Satyanarayan]

- Challenges in mobile computing with focus on the need for adaptation based on variable connectivity
- Challenges are still an issue today!

Four Constraints

- Resource Poor Elements
 - Must carry around, less weight, less power, small size, and ergonomics exacts penalty:
 - Lower CPU speeds, memory size, and disk size
 - Always resource poor relative to static elements
- Inherent Hazards
 - People can interfere with communication, leave / misplace / steal device, easily damaged
- Variable Connectivity
 - Bandwidth and latency highly dependent upon environment. i.e. gaps in coverage, buildings around, etc.

Four Constraints (cont'd)

- Finite Energy Source
 - Different from “resource poor”
 - Energy is consumed, CPU, weight, disk size, and others are just constraints
 - Other resources follow Moore's law, while battery technology doubles 5-10 years, well below the Moore's technological growth curve
- *These constraints are not artifacts, but intrinsic to mobility ... always will be there*

Need for Adaptation

- Mobility exacerbates the tension between autonomy and interdependence that is characteristic of all distributed systems?
- Limited resources argue for reliance on static servers
- Variable connectivity argues for autonomy
 - Want to perform useful work while disconnected, classic example is taking laptop on vacation or airplane
- Connectivity is low but laptop must adapt and allow user to do useful work for upcoming presentation/meeting
- Must strike a balance between these tension!
- Balance must be sensitive to circumstance
- *“Mobile clients must be adaptive” three strategies*

Strategy #1

- Application is responsible (no system changes)
 - Laissez-faire policy, application handles adaptation
 - Advantage is no new system support is needed
 - Disadvantage is duplication of work, and no arbitrator exists to resolve incompatible resource demands of competing applications
 - Nothing can enforce limits on resource usage

Strategy #2

- System is responsible (no application changes)
 - Application transparent policy
 - Advantage is that it is backwards compatible with existing applications, but there exist cases in which adaptation by the system is useless or even counter-productive

Strategy #3

- Responsibility is shared (both change)
 - This approach tries to marry the benefits of the previous two
 - The system can still arbitrate resources between competing applications, but it is up to the application to determine how best to adapt in terms of the resources offered by the system

Impact on Classic Client / Server

- Coping with the constraints of mobility requires us to rethink classic client/server
- Resource limitations of clients require operations normally performed to be offloaded to resource rich servers
- Because we need to hide the latency, low bandwidth, and variability of mobile link, clients sometimes need to emulate the functions of the server
- *These are short term deviations for performance and availability*
 - *In the long term perspective, still is the classical client/server model*

CODA Experiment: (Application-Transparent)

- Disconnected operation
- Optimistic replication
- Support for weak connectivity
- Isolation-only transactions
- Server replication
- *A distributed filesystem for networks with variable connectivity*
 - More on this in the next paper

Odyssey Experiment: (Collaborative)

- Later (1995–1999) platform for mobile computing
- Experiments in application-aware adaptation
- *More on this in future presentations*

Five Topics for Exploration

- Caching metrics
- Semantic callbacks and validators
- Algorithms for resource revocation
- Analysis of adaptation
- Global estimation from local observations

Caching Metrics (Issues)

- Today, only metric is miss ratio
 - Fails to account for timing
 - Cost of miss is non-uniform
 - Ex: A man page miss is much less harmful than miss on a critical e-mail app module when I'm sending critical mail about missing a meeting, while on the way
 - *Disconnected miss penalty can be very high*
- New metrics should be:
 - Consistent with qualitative perceptions
 - Computationally simple and measurable

Caching Metrics (Open Questions)

- Appropriate set of caching metrics?
- Circumstances to use these metrics?
- Monitoring these metrics efficiently?
- Implications on caching algorithms?

Semantic Callbacks and Validators (Issues)

- Client obtains object satisfying predicate P from server
- Server remembers predicate Q which is much cheaper to compute and Q implies P
- If Q true on server $\rightarrow P$ valid on client
- On update if Q becomes false, server notifies client that object may be stale
- On next access the client must contact server and obtain fresh copy satisfying P
 - Like GET-IF-MODIFIED HTTP request, or in Coda, version on an object (P) versus version on volume containing object (Q)
- This is useful not only for mobile computing, but also in the wide-area network case as well

Semantic Callbacks and Validators (Open Questions)

- When are they (not) useful?
 - These Q's are inefficient when the state is changing quickly, too many unnecessary validations occur
- Forms for P and Q for common data?
- How to generate Q from P quickly?
- Cost/benefit trade-off?

Algorithms for Resource Revocation (Issues)

- Some applications are more important
- Cost for same revocation may differ among applications
- Not common in current OS's
- *Application-aware adaptation complicates the problem!*

Algorithms for Resource Revocation (Open Questions)

- Problem Formulation?
- Characterize differential impact on applications?
- Simultaneous multi-resource revocation strategies?
- Distinguishing between recovery efforts?
- How to handle deadlocks?

Analysis of Adaptation (Issues)

- Agility is the ability of the client to promptly respond to perturbations
- Agility for comparison of adaptation
- Agility with respect to variables
- Increased agility may lead to instability
 - I.e., the client spends all of its time adapting

Analysis of Adaptation (Open Questions)

- Appropriate metrics of agility?
- Systematic techniques to improve agility?
- Appropriate metrics of stability?
- Can stability be insured by design?

Global Estimation from Local Observations (Issues)

- Adaptation requires awareness of circumstance
- Rely on local observations
- Interpret these observations in context
- From observations must make global estimates
 - Signal strength, packet rate, avg RTT, dispersion in RTT are local estimates
 - Changes in these can be due to many non-local phenomena: from congestion on an intermediate server to interference from weather
- So far, there is no systematic theory to guide global estimation

Global Estimation from Local Observations (Open Questions)

- What are the appropriate ways?
- Can the estimate errors be bound?
- Global estimates relation to agility?
- Support for improved estimation?
- Out of band channels?
- Quantify the benefits?

Conclusions

- Mobility exacerbate tension between autonomy and interdependence
- Adapt by trading compute resources for communication
- Approaches are viable!
 - CODA confirms application-transparent adaptation to be viable
 - Initial work with Odyssey suggests application-aware adaptation is viable

5-minute Break

A Highly Available File System for a Distributed Workstation Environment [Satyanarayan]

- DFS: Shared large-scale distrib. computing environ.
- Problems:
 - Scalability → push tasks on clients
 - Consistency (concurrency control)
 - Handling partitions/disconnections
 - Availability
- Must choose from the above set
 - Cannot have all simultaneously. *Why?*
- Design Goals:
 - Availability, Performance, Consistency

Design Goal: Availability

- How to ensure the clients access to the files they want at all times?
 - Dealing with partitions ? Server Replication
 - Dealing with disconnections (harder) ? Disconnected Operation

Design Goal: Consistency

- What happens when a file is accessed by two different clients? → maintain consistency
 - Shared/exclusive locks (pessimistic, low availability)
 - Leases (temporal locks)
 - Laissez-faire (optimistic, less consistency)
- Key Idea: be optimistic and worry about problems only when they occur
- The assumption made by Coda is people are unlikely to be concurrently modifying a file, so they choose the optimistic policy

Architecture

- Elements
 - Lots of untrusted clients
 - A few trusted servers forming a core
 - Communications through RPC with encryption and authentication
- Mechanism
 - Servers store files (first class copies)
 - Clients cache them for performance and availability (second class copies)
- Cache agent (Venus)
 - Handles remote file system requests
 - Caches files in local disk
 - Minimum unit is a file (makes it simple)
 - Masks disk operation

Server Replication

- Needed to support partitions
 - Minimum unit of replication:
 - Volume (aggregate of files in the same subtree)
 - Using volume implies less state to handle for validation
- Read One, Write All (only on close)
 - Clients?
 - Coherence is achieved through callbacks from the server to the client
 - Servers?
 - Consistency triggered by client access

Disconnected Operation

- Needed to support disconnections,
 - Where the applications still want to see the file system as if it were composed of ALL the files
- Solution:
 - When connected, try to guess which files are going to be used when disconnected and fetch them
 - Store the changes made while disconnected and reintegrate them when connecting
 - Differences or conflicts in files that cannot be resolved are punted to the user

Disconnected Operation (cont'd)

- Key idea:
 - Users are good at identifying their long-term file needs
 - LRU is a good estimate of the user's short-term file needs
 - Disconnected Operation is possible

Not Quite!

- Example: You're working for a long time
- LRU replaces initialization files which you have not touched in a while
- Now, your system crashes and you reboot: but you can't!
 - Your system files were punted by LRU!

Hoarding

- Prioritized algorithm
 - Implicit info (recent history)
 - While you work peek at the files you touch and guess at which ones you might need
 - Explicit info (hoarding profiles)
 - You definitely need some files: gcc, loader, system kernel, these are “pinned” or kept with you at all times
- Hoard walking (restores equilibrium)
 - Adds new children in subdirectories
 - Reevaluates priorities
 - Resolves callback breaks

Emulation

- Read: look in the Cache
 - If miss, block process or return error code
- Write: store changes in a per-volume log (replay log)
 - Use refined policies to make it small: log compression, etc.
- Cache management: Don't free updated objects (cache management)
 - Persistence: store metadata in RVM, and data in Unix files
 - RVM (non-volatile storage) permits trading off performance for commitment (bounded persistence)

Reintegration

- Replay algorithm
 - Sending log to server
 - Can be incremental
 - As connectivity comes and goes
 - Use application-specific resolvers to handle conflicts
 - Avoids bothering user and cascading aborts
- If all fails, ask the user
 - Least desirable choice

Performance (1991)

- Reintegration after some hours:
 - Takes 1 minute at client
- Cache size needed:
 - 60 MB for one day (but heavily workload dependent)
- Likelihood of W/W conflicts: 0.4% in a week (encourages optimistic replication)
 - Depends on workload (e.g., shared paper)

DBMate [Badrinath]

- Mobile Databases
 - Small clients
 - PDAs, laptops, smart phones
 - Lots of semantics in data, unlike files
 - Location (database for S.F.)
 - Type (database for sales, salaries, etc.)
 - Need to support disconnected operation
- DBMate exploits locality by using database semantics to carry only locally relevant data in database clients
 - Ex: A client downloads all records that correspond to Menlo Park

New Workload Characteristics

- Clients hoard and later reintegrate, just like in Coda, but...
 - Bursty I/O
 - LRU cache becomes cold for hoarders
- Clients workload can be profiled
- Weak consistency and small footprint

Related Projects

- CODA
 - Only write/write conflicts detected (DBs need read/write also)
 - Server organization is almost unchanged from previous AFS
- RUMOR
 - Peer-to-peer synchronization
- CODA is insufficient for disconnected database applications
 - Has file granularity
 - Relational tables (database tables) >>> files.
 - Map each tuple to file → file explosion problem
 - Map relation to file → files are too big and how do you keep integrity constraints between relations, how do you map entire schema to files?
 - Need to handle R/W conflicts for transaction atomicity
 - Must treat relations differently from files!

Replication vs Hoarding

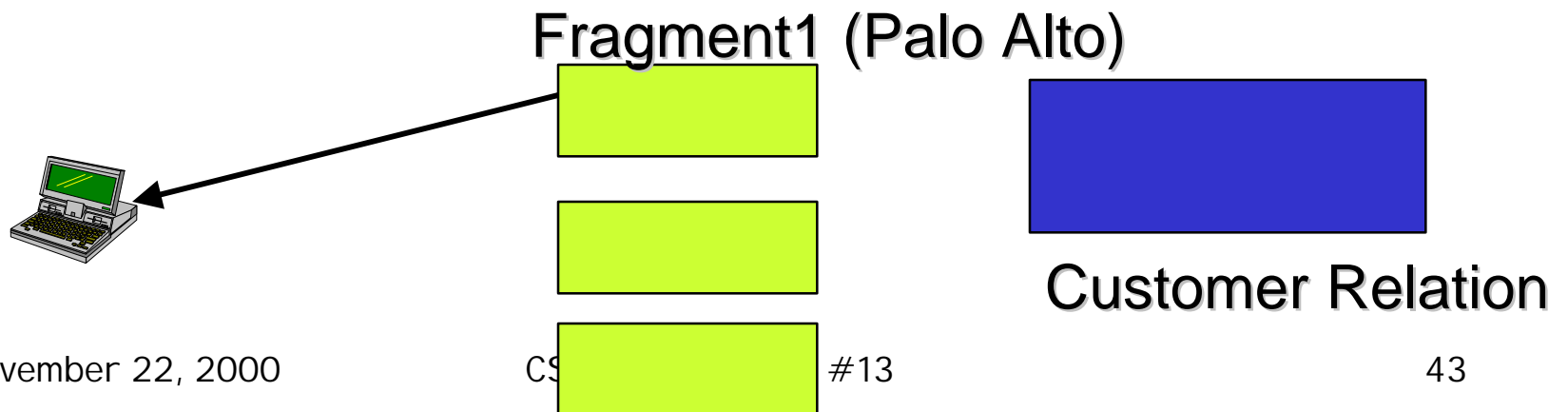
- Replicas are identical copies
 - Hoarding is more like caching appropriate workload data
- Assumes persistent connection, any disconnection is due to failure
 - Hoarding=>Disconnection is more deliberate
 - Prepare for it
 - Not like a local cache, in local cache, servers still process commits, service cache misses

Design Philosophy

- Consider two distinct workloads
 - Hoarders and traditional transactions
 - Bursty I/O, locality, impact on caching
- Support disconnected operation in DBs
 - Allow local commits and later reintegration
- Support a consistency model for disconnected mobile clients
 - Allow application specific conflict resolution

Hoarding Granularity

- Mobile clients hoard fragments of relations
 - In between a table and a tuple
- Fragments need to capture locality of access while disconnected
 - Download Palo Alto customers (day's workload)



Characterizing Fragments

- Mobile clients download fragments and later reintegrate
 - We need efficient ways to create and manage fragments
- Define hoard attributes (attribute of a relation)
- Fragments are created by specifying qualifiers on hoard attributes

Example of hoard attribute

CUSTOMER(CNUM,LOC,NAME,NETWORTH)

PRIMARY KEY IS CNUM

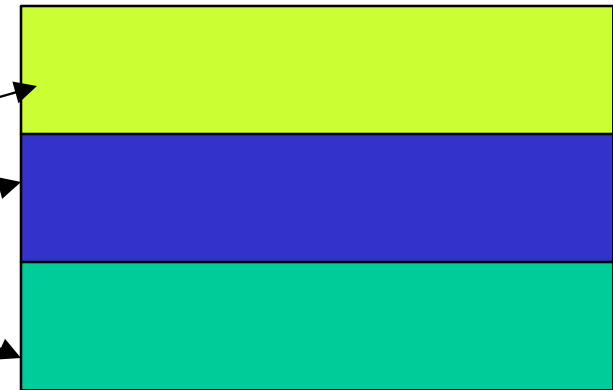
HOARD ATTRIBUTE IS LOCATION

Qualifiers for location

Q1: Location = ``Palo alto``

Q2: Location = ``Berkeley``

Q3: Location = ``Menlo park``



Physical Organization

- Fragments represent partitions of a database
 - Do we want to affect physical organization?
- Logical and physical hoard attributes
- Declare hoard attribute as physical
 - Fragments are physical fragments on disk
- Declare hoard attribute as logical
 - Index provided for accessing the fragment

Choices

- All hoard attributes are logical
 - Create a separate index for each of the logical fragments
 - No impact on traditional, slows mobile
- All hoard attributes are physical
 - Reorganize data on disk
 - Impact on traditional, Fast for mobiles
- Some are physical, some are logical
 - Reorganize only fragments corresponding to physical hoard attribute

Consistency model

- Detect conflicts at the tuple level
 - Modified Davidson algorithm
- Reintegrating transactions serialized after any transactions at the server
- Determine dirty set and reject a reintegrating txn if $TS_w(x) > TS_H(C)$
- else commit T after writing `WRITESET(T)`

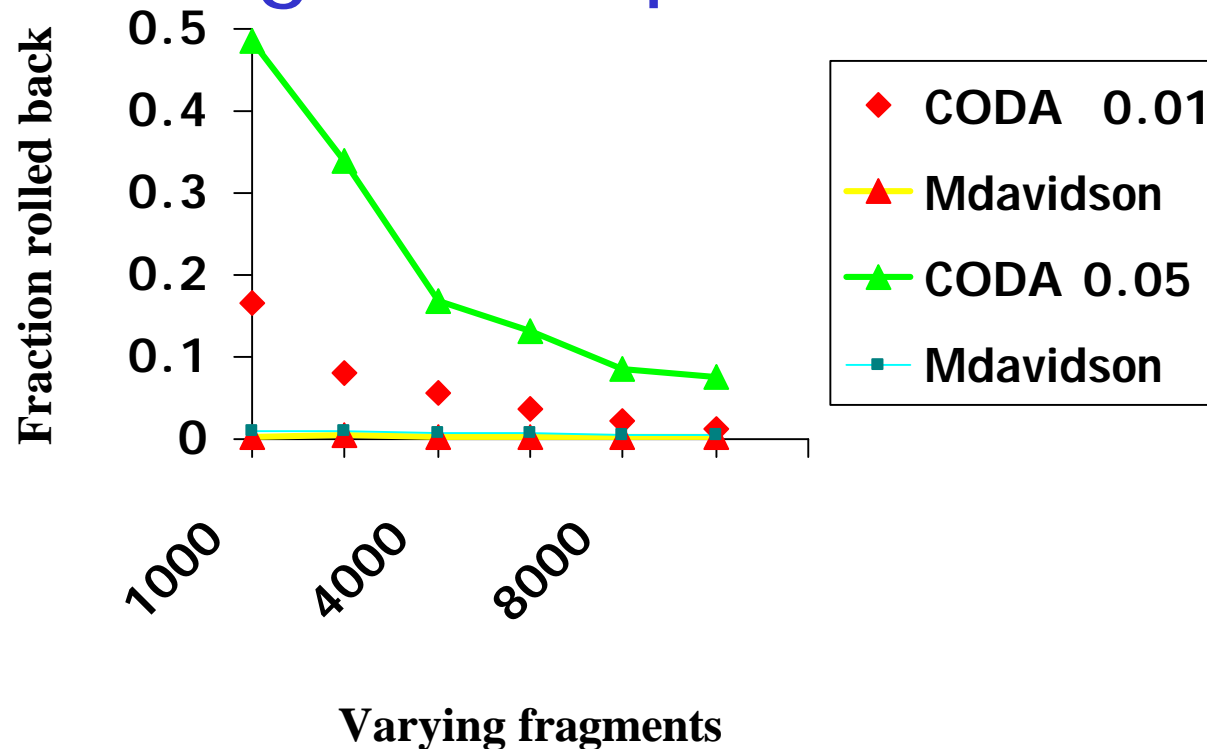
Simulation model

- Traditional transactions, hoarding transactions
- Mean number of tuples accessed/hoarded
- Average disconnection time = 2 hrs
- Fragments are 80% physical
- DB size 40K pages, size of each page 4K, 40 tuples/page

Increasing # of Fragments

- Lower traditional performance, better hoard
- Consecutive tuples spread over fragments → more disk accesses, but clustering of fragments improves hoard performance

Reintegration performance



- CODA-style benefits from smaller fragments reducing collision cross-section

Conclusions

- Designed support for disconnected operation in databases
- Proposed fragments as a unit of granularity for hoarding
- Physical and logical hoard attributes for organizing server data
- Modified consistency model

Discussion: Could You Implement CODA on DBMate or vice-versa

- Is there a common set of functionality (primitives) upon which both could be implemented efficiently?
- Badrinath's Point of View:
 - Databases and file systems are two different creatures and must be treated differently
 - In short, no, unless you are really clever; otherwise, the system becomes too complex

Another Point of View

- Filesystems and databases are coming together
 - Want same transactional atomicity and consistency properties that databases traditionally provide for filesystems
 - For example, this is useful when writing a source code control system that works correctly under long periods of disconnected use
 - It is nice to have a set of files commit to the repository or not without having the system developer to figure out how to undo all committed files if one in the set cannot commit due to a conflict
- Is there any common ground between file systems and database upon which both could be built for efficient use in the mobile environment?