Novel File Systems: The Evolution of Coda

Presented by Hakim Weatherspoon

M. Satyanarayanan

- Systems faculty at Carnegie-Mellon University
- Andrew Project
  - Distributed computing environment begun in 1983
  - IT joint venture between CMU and IBM
  - Focused on workstations: client-server

- Lead Andrew File System
- Inspired CODA and another 20 years of research

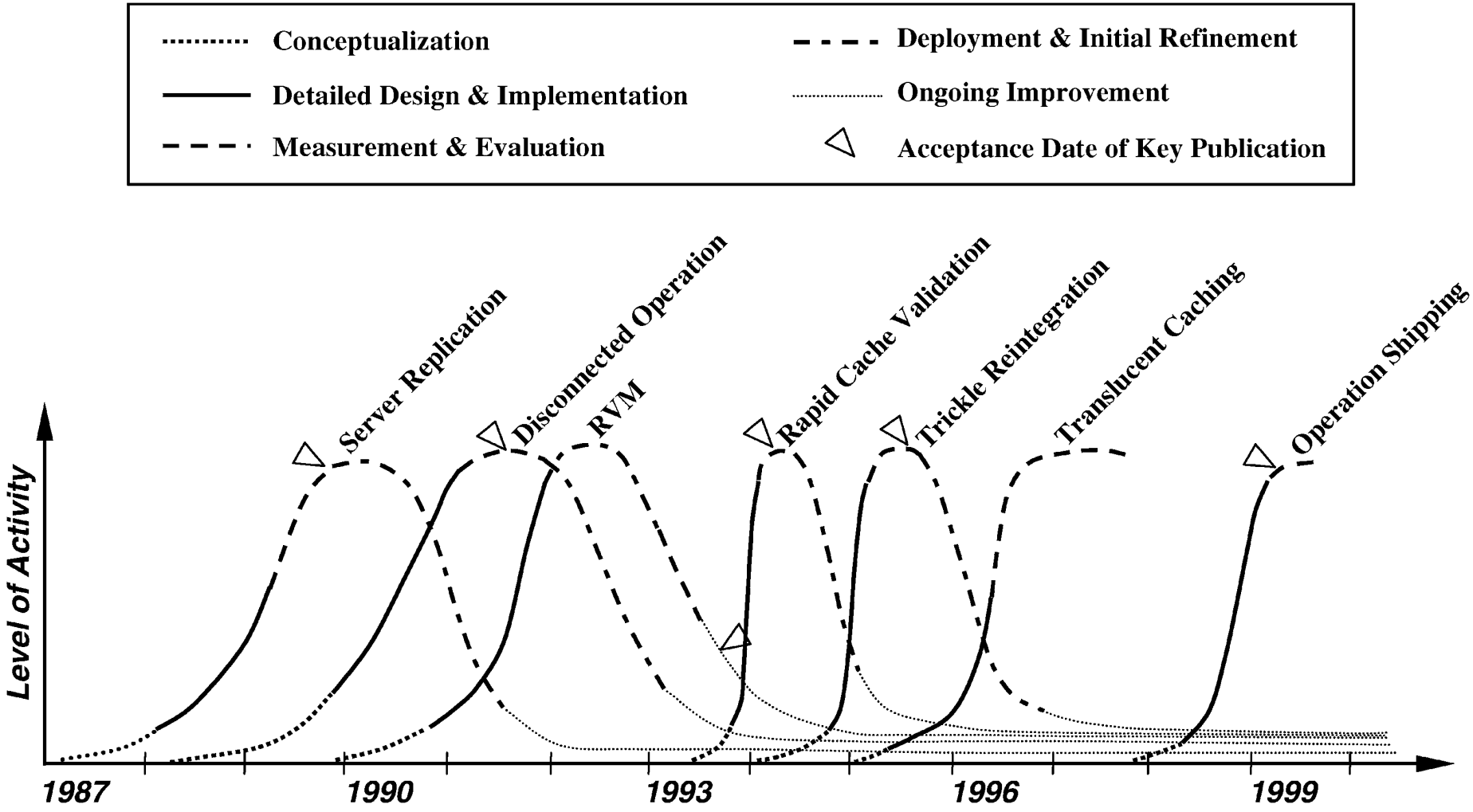
Paper overview

- Reviews the multiple contributions of Coda:
  - Optimistic replication
  - Trickle reintegration to support weakly connected workstations
  - Isolation-only transactions
  - Operation shipping
- Ends with a few lessons learned

Motivation for Coda

- Epilogue to the Andrew File System (AFS)
- AFS was found to be vulnerable to ***server and network failures***
  - Not that different from NFS
  - Limits scalability of AFS
- Coda addresses these problems through ***optimistic replication***

Timeline


Conceptualization
Detailed Design & Implementation
Measurement & Evaluation
Deployment & Initial Refinement
Ongoing Improvement
Acceptance Date of Key Publication
Level of Activity
Server Replication
Disconnected Operation
RVM
Rapid Cache Validation
Trickle Reintegration
Translucent Caching
Operation Shipping
1987
1990
1993
1996
1999

Lessons Learned from 20 years of Coda

- Optimistic replication can work
  - Must use for performance
- Real systems research needs
  - Real system artifacts
  - Real users
- Timing
  - Need to be lucky
- Research vs product
  - Long ‘product’ tail
- Moores law
  - Worked then. Does it work now?

Lessons Learned from 20 years of Coda

- Code reuse is a double edged sword
  - Good initially, but locks you into a particular regime
- Need system admins
  - Deeply held secret
- Small projects never die
  - Also small features hard to remove

Server Replication: 1987-1991

- ***Optimistic replication control protocols*** allow access in disconnected mode
  - Tolerate temporary inconsistencies
  - Promise to detect them later
  - Provide ***much higher data availability***
- Optimistic replication control requires a reliable tool for detecting inconsistencies among replicas
  - Better than LOCUS tool

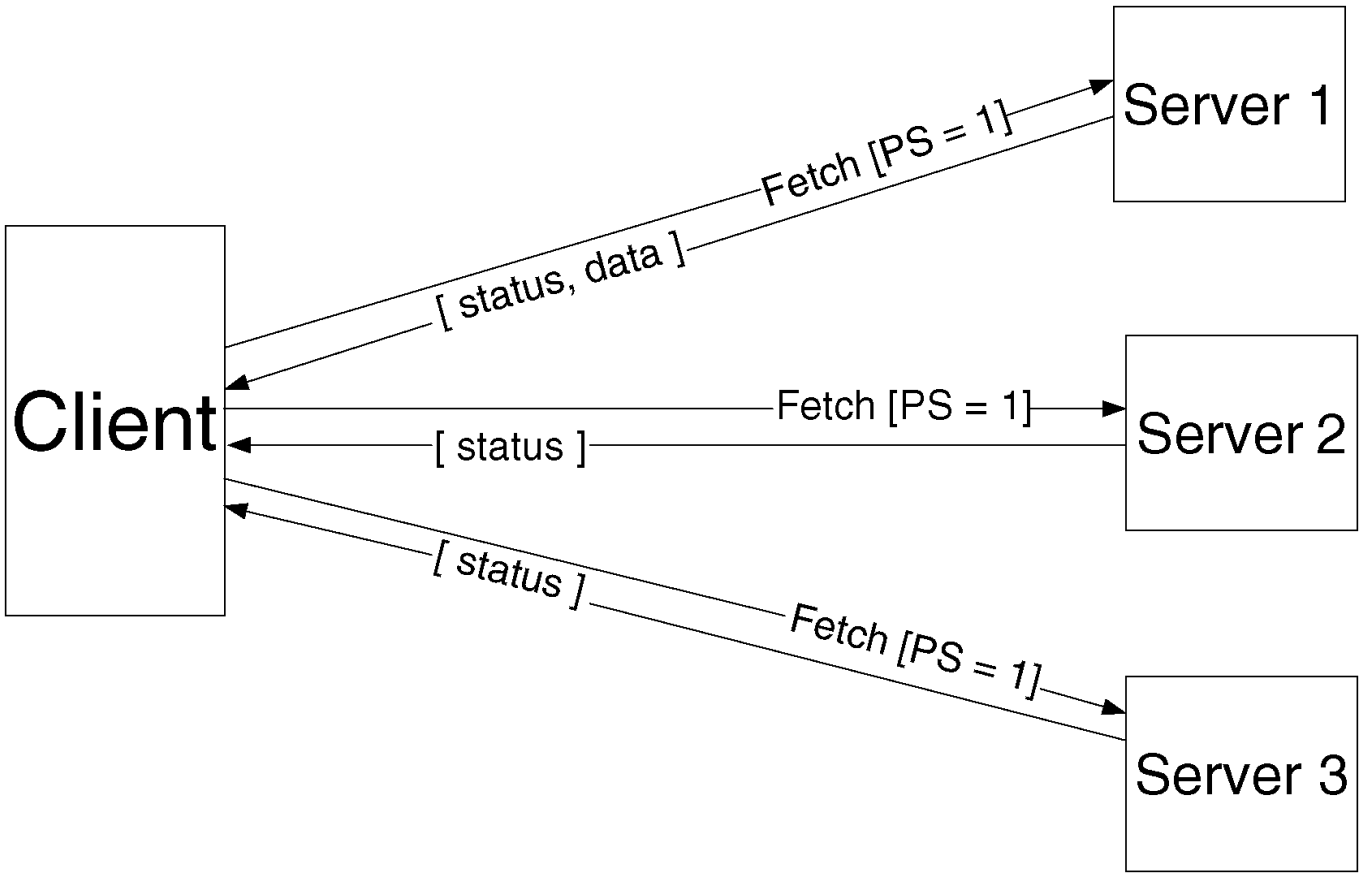
Server Replication

- Unit of replication is ***volume*** (subtree of files)
- Set of servers containing replicas of a volume is ***volume storage group*** (VSG)
- Currently accessible subset of VSG is ***accessible volume storage group*** (AVSG)
  - Tracked by cache manager of client (***Venus***):

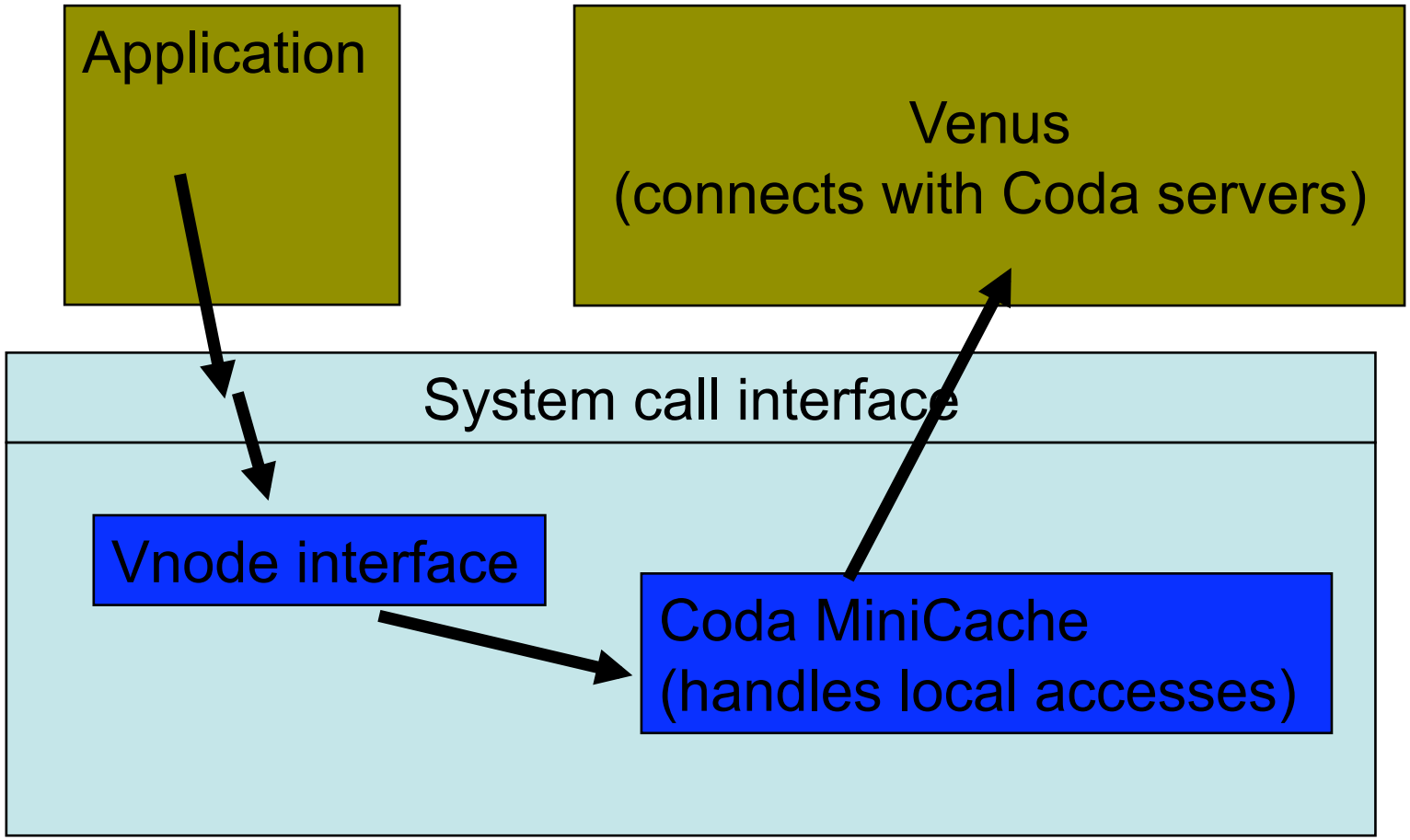
Read protocol

- ***Read-one-data, read-all-status, write-all***
- Each client
  - Has a ***preferred server (VS)***
  - Still checks with other servers to find which one has the latest version of a file
- Reads are aborted if a conflict is detected
- Otherwise a callback is established with all servers in AVSG

Read protocol



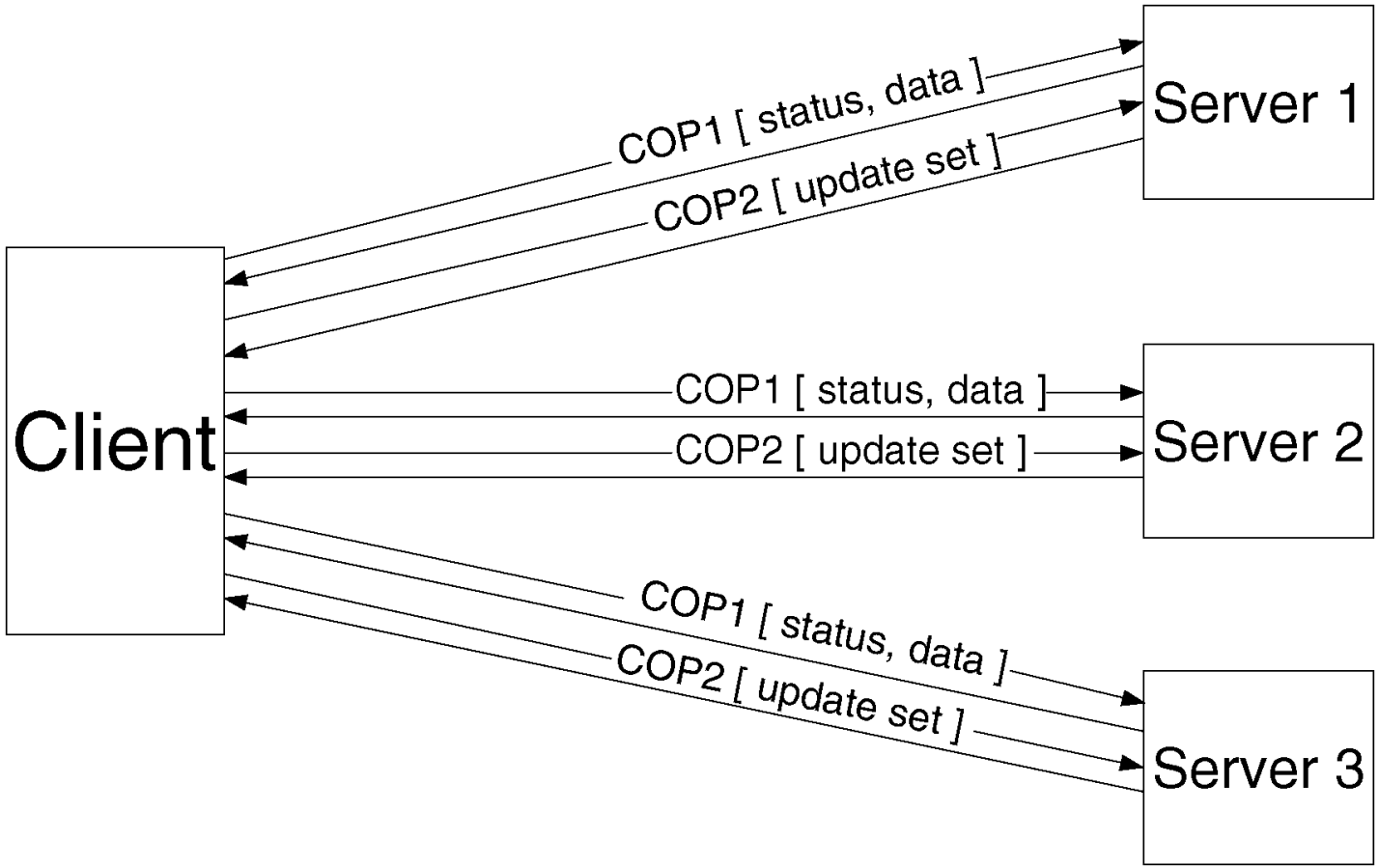
Client Structure



Update protocol

- When a file is closed after modification, updated file is transferred in parallel to all members of the AVSG
- Directory updates are also written through to all members of AVSG
- Coda checks for replica divergence before and after each update
- Update protocol is ***non-blocking***

Update protocol



Consistency model

- Client keeps track of subset ***s*** of servers it was able to connect the last time it tried
- Updates ***s*** at least every τ seconds
- At **open time**, client ***checks*** it has the ***most recent copy*** of file among all servers in ***s***
  - Guarantee weakened by use of ***callbacks***
  - Cached copy can be ***up to*** τ ***minutes behind*** the server copy

Fault-tolerance

- Correctness of update protocol requires atomicity and permanence of metadata updates
- Used first Camelot transaction management system:
  - Too slow and Mach-specific
- Coda uses instead its own ***recoverable virtual memory*** (RVM)
  - Implemented as a library

LRVM

- Thesis
  - *RVM ... poses and answers the question "What is the simplest realization of essential transactional properties for the average application?" By doing so, it makes transactions accessible to applications that have hitherto balked at the baggage that comes with sophisticated transactional facilities.*
- Answer
  - Library implementing No-Steal, No-Force virtual memory persistence, with manual copy-on-write and redo-only logs.

LRVM

- Goal
  - allow Unix applications to manipulate persistent data structures (such as the meta data for a file system) in a manner that has clear-cut failure semantics.
- Existing Solutions
  - Camelot too heavyweight
  - Wanted “lite” solution
    - Do not provide (unneeded) support for distributed and nested transactions, shared logs, etc.
- Proposed Solution
  - Library that provides only recoverable virtual memory

LRVM: Lessons from Camelot

- Overhead significant
  - multiple address spaces
  - constant IPC between them
- programming constraints
  - Heavyweight facilities impose programming constraints.
- Size and complexity
  - Camelot too big
  - Too dependent on Mach
  - maintenance headaches and lack of portability

LRVM: Lessons from Camelot

- Camelot had a object and process model.
  - Its componentization led to lots of IPC.
  - It had poorly tuned log truncation.
  - Was perhaps too much of an embrace of Mach.
- However, a lot of good learned from Camelot
  - the golden age of CMU Systems learned a lot from the sharing of artifacts: Mach, AFS, Coda...
  - A lot of positive spirit in this paper.

LRVM: Architecture

- Only addresses the problem of Recovery.
- Stores Virtual memory in external data segments found in stable storage.
- Portable with a library that is linked in with applications.
- “Value simplicity over generality” by adopting a layered approach.
- Provides independent control over atomicity and concurrency as well as other problems such as deadlocks and starvations.

Layered Approach of RVM

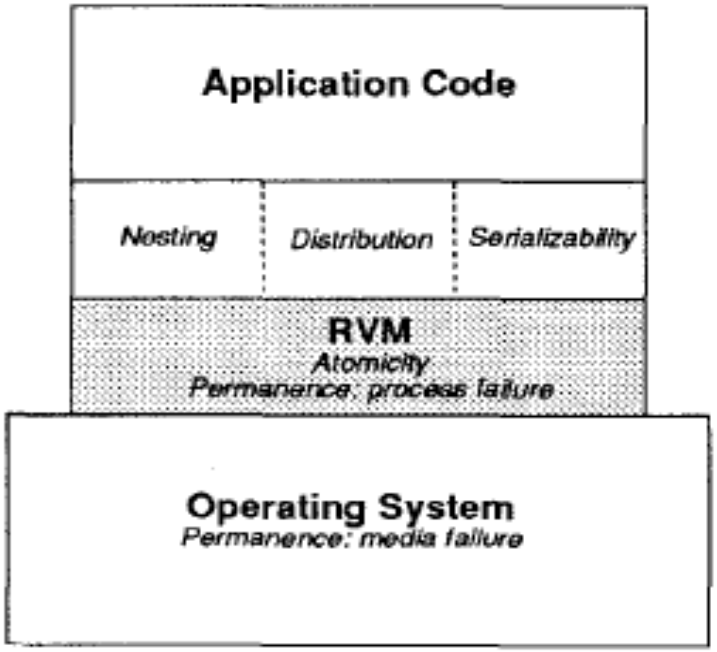
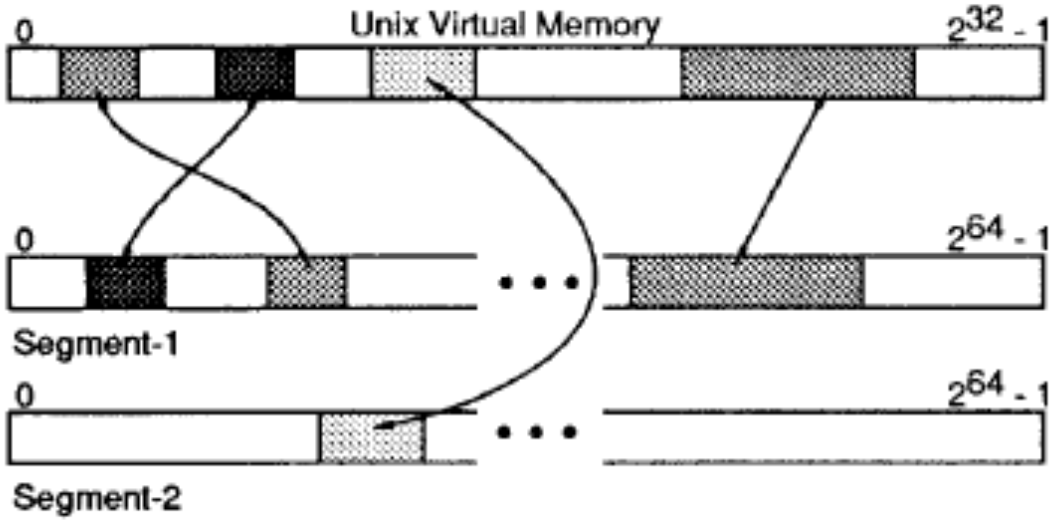


Figure 2: Layering of Functionality in RVM

LRVM: Segments and Regions

- Applications map regions of segments into their virtual memory.



LRVM: Sequence of Operations

- Select regions in virtual memory to be mapped.
- Get a global transaction ID.
- Successful commit saves segments in log.

LRVM: Crash Recovery

- Recovery consists of reading the log from tail to head and then reconstructing the last committed changes.
- Modifications are applied to the external data segment.
- Log is emptied.

LRVM: Truncation

- Reclaiming space in the log by applying changes to the external data segment.
- Necessary because space is finite.

LRVM: Performance

- Beats Camelot across the board.
- Lack of integration with VM does not appear to be a significant problem as long as ration of Real/Physical memory doesn't grow too large.
- Log traffic optimizations provide significant (though not multiple factors) savings.

LRVM: Summary

- RVM addresses only the problem of recovery in VM and introduces a “neat” layered structure to address the other problems
- manipulate persistent data structures
  - In manner that has clear-cut failure semantics
- Experience
  - Heavyweight fully general transaction support facility led to lightweight facility that only provides recoverable virtual mem
- However,
  - Paper did not show that layered approach can perform well
- Lesson
  - Bulding OS, do few things well instead of being general

Disconnected Operation:1988-1993

- Started as tool allowing a client isolated by a network failure to continue to operate
- Made possible thanks to
  - Optimistic philosophy
  - ***File hoarding*** in client cache
- Gained importance with arrival of portable computers
  - Resulted in ***voluntary disconnections***

Disconnected Operation

- **File Hoarding:**
  - Coda allows user to specify which files should always remain cached on her workstation and to assign priorities to these files
- When workstation gets reconnected, Coda initiates a ***reintegration process***
  - Changes are propagated and inconsistencies detected

Disconnected Operation

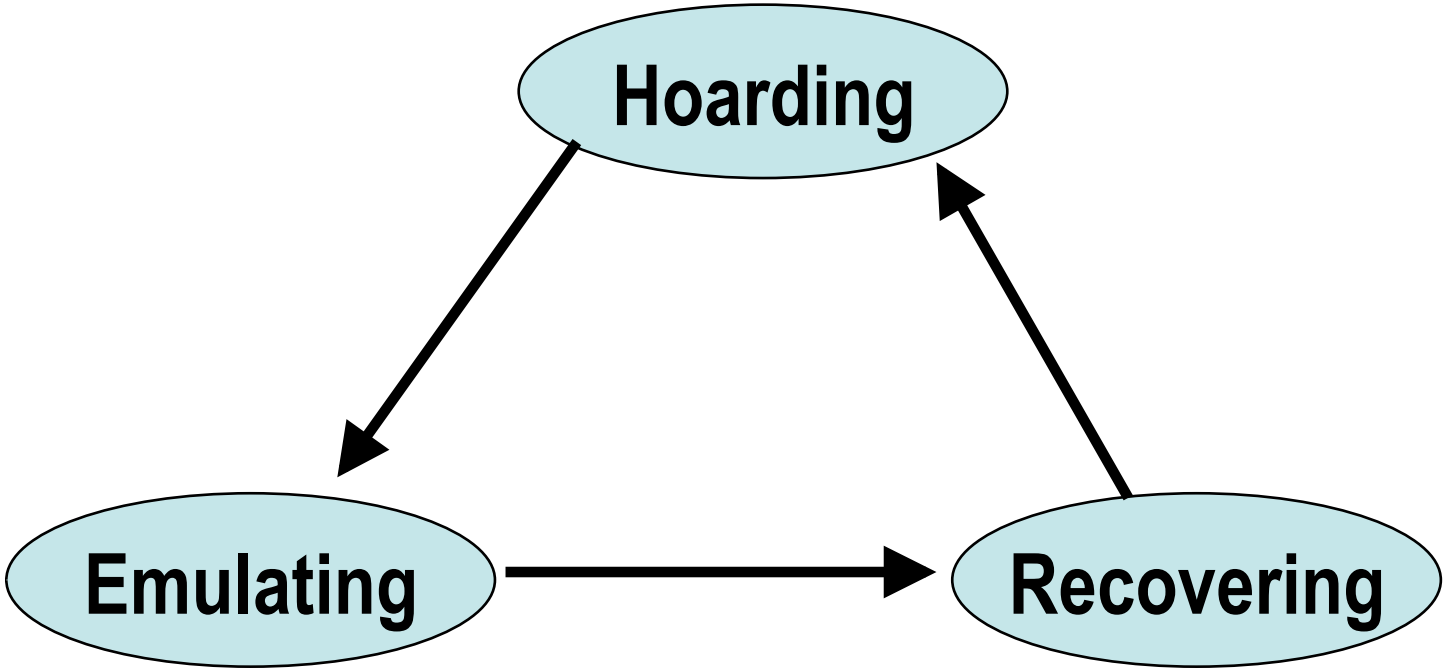
- Disconnected operation mode complements but does not replace server replication
  - Cached replicas are only available when client workstation is turned on
  - Make server replicas ***primary replicas*** and cached replicas ***secondary replicas***

Implementation

- Three states:

1. ***Hoarding:***
   Normal operation mode
2. ***Emulating:***
   Disconnected operation mode
3. ***Reintegrating:***
   Propagates changes and detects inconsistencies

Implementation



Implementation

- Coda maintains a per-client hoard database (HDB) specifying files to be cached on client workstation
  - Client can modify HDB and even set up hoard profiles

Implementation

- In disconnected mode:
  - Attempts to access files that are not in the client caches appear as failures to application
  - All changes are written in a persistent log, the ***client modification log*** (CML)
  - Venus removes from log all obsolete entries like those pertaining to files that have been deleted

Conflict Resolution: 1988-1995

- Coda provides automatic resolution of simple directory update conflicts
- Other conflicts are to be resolved manually by the user

Objectives

- *No updates should ever be lost without explicit user approval:* conflicts must be detected
  - Do they ensure this?
- *The common case of no conflict should be fast*
  - Is it?
- *Conflicts are ultimately an application-specific concept:* think of updates to a schedule
- *The buck stops with the user:* automatic conflict resolution cannot solve all problems

Approaches to conflict resolution

- ***Syntactic approach:***
  - Uses version information
  - Fast and efficient
  - Weak in their ability to resolve conflict
- ***Semantic approach:***
  - Slower but more powerful

Coda solution

- Coda uses
  - ***Syntactic approach*** to detect absence of conflicts
  - ***Semantic approach*** to resolve possible conflicts

Directory conflict resolution

- Always automatic
- Uses a **log-based**
- Two cases to consider
  - After disconnected operation
  - Across conflicting replicas

After disconnected operation

- Each server tries to apply the ***client modification log*** (CML) send by the client during reintegration
- If this attempt fails, client directory is marked in conflict.

Across divergent replicas

- Each server replicas of a volume has a **resolution log** containing entire list of directory operations
  - In reality, it is frequently truncated
  - Remains almost empty when there are no failures
- Recovery protocol locks the replicas merges the logs and distributes the merged logs.

Other solutions

- Must keep track of partial deletes:
  - If one of the two replicas has a directory A, does it correspond to a file
    1. **recently created**, or
    2. **recently deleted**.
- Must keep **ghost entries** for directory entries that were recently removed
  - Hard to know when these entries can be purged

Application-Specific File Resolution

- Entirely done at client

Conflict representation

- Coda displays read-only versions of inconsistent objects

Frequency of conflicts

- Probability of two different users modifying the same object less than a day apart is less than 0.0075

Weakly Connected Operation: 1993-1996

- Broad principles
  - Do not punish strongly connected clients
  - Do not make life worse when disconnected
  - Do it in the background if you can
- Rapid validation of cache
- Trickle Reintegration

Lessons Learned from 20 years of Coda

- Optimistic replication can work
  - Must use for performance
- Real systems research needs
  - Real system artifacts
  - Real users
- Timing
  - Need to be lucky
- Research vs product
  - Long ‘product’ tail
- Moores law
  - Worked then. Does it work now?

Lessons Learned from 20 years of Coda

- Code reuse is a double edged sword
  - Good initially, but locks you into a particular regime
- Need system admins
  - Deeply held secret
- Small projects never die
  - Also small features hard to remove

Next Time

- Read and write review:
- Do Lab 1 due tomorrow
- Project Mtg today at 2:30pm in Systems Lab
- Check website for updated schedule