
Synchronization of Distributed Data

Nicu D. Cornea

- **Disconnected Operation in the Coda File System**

Even in distributed systems, the failure of a critical point might inconvenience the users

- It would be nice to benefit from the use of a shared repository, but to be able to work even if that repository becomes inaccessible.

- * **Caching of data**

- Widely used to improve performance
 - Can be used to enhance **availability**

- **Exploiting Weak Connectivity for Mobile File Access**

- Goal: to preserve **usability** even when dealing with weak connectivity

Coda File System

- Distributed File System, developed at Carnegie Mellon University in Pittsburgh
- Client view:
 - Single, location-transparent shared Unix file system
- Namespace mapped to individual file servers at the granularity of sub-trees called **volumes**
 - Volume Storage Group (**VSG**) = set of replication sites for a volume
 - Available VSG (**AVSG**) = VSG visible to a client
- High availability achieved by:
 - Server replication
 - * Local caching
 - * Cache coherence protocol based on **callbacks**
 - * Modifications propagated in parallel to all AVSG sites
 - **Disconnected operation** (AVSG becomes empty)
 - * Cache manager (Venus) services FS calls using cached files
 - * Upon reconnection, propagates the updates

Design Considerations

- Scalability
 - Callback based cache coherence
 - Whole-file caching
 - * Simpler failure model: cache miss can only happen on “open”
 - Smart clients
- Portable workstations
 - Most frequent users of disconnected operation
 - Voluntary/Involuntary disconnection handled by the same mechanism
- First vs. Second class replication
 - First class replicas – replicas residing on servers
 - * Persistent, widely known, secure, available, complete, accurate
 - Second class replicas – cached copies at clients
 - * Periodically revalidated with respect to a first-class replica
 - * Performance, scalability

Design Considerations (2)

- Replica Control (**Optimistic** vs. Pessimistic)
 - Pessimistic
 - * Client has to obtain shared or exclusive control over a file before disconnection and retain it until reconnection
 - * Meanwhile all other clients are blocked from reading/writing that file
 - Optimistic
 - * Allows changes to be made anywhere and deals with conflicts as they arise
 - * Client's updates are immediately visible to it's accessible universe unless there is a conflict
 - * The system is more complicated
 - * Choice based on the observation that conflicts are rare

Client Structure

- Cache manager: Venus
 - Intercepts Unix FS calls via Vnode interface
 - Mini cached used to improve performance by handling calls that can be serviced without remote access
 - States:
 - * Hoarding
 - * Emulation
 - * Reintegration

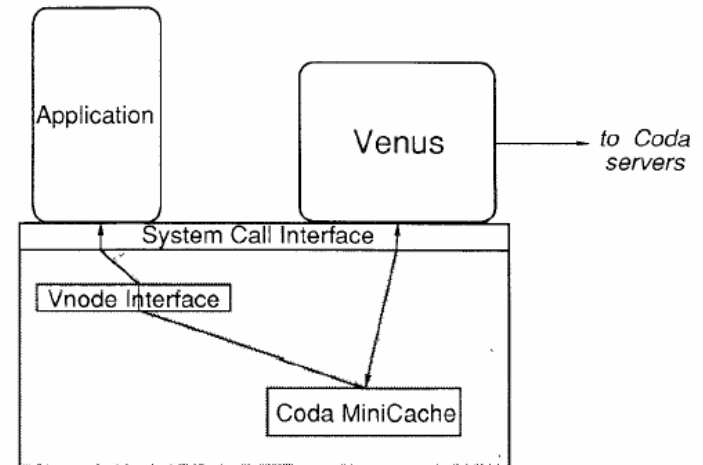
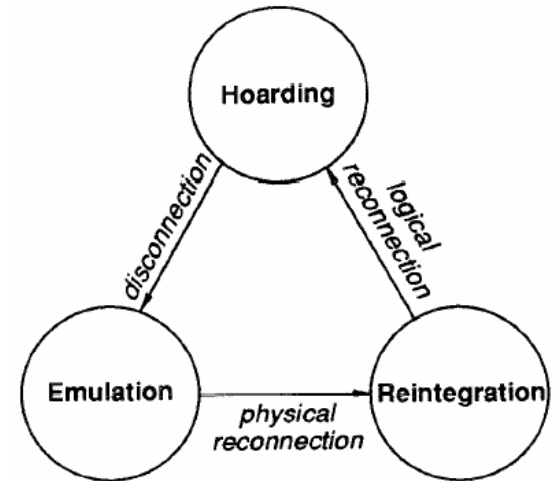


Fig. 2. Structure of a Coda client.



Hoarding

- Caches useful data in anticipation of disconnection
- Prioritized Cache management
 - * Usage history
 - * Hoard database (HDB)
 - User can specify objects of interest that should be cached in the event of a disconnection
 - Priority associated with objects
 - * Final priority of an object – combination between the priority in HDB and the usage history
- Hoard walk – periodically restores cache equilibrium
 - * No un-cached object should have higher priority than a cached one
 - * Every 10 minutes or at user request

Emulation

- Disconnected operation
- Functions as a pseudo-server
 - Updates accepted by Venus have to be eventually validated by a server
 - Returns error code on cache miss
- Update information is logged (replay log) so that it can be played back when reintegrating
 - Replay log is played back by the servers
- When cache space runs out ... 😊

Reintegration

- Changes made during Emulation are propagated to the servers
- Per volume
- Replay log is sent to the servers
- Conflict handling:
 - Every object has an associated version
 - If version from replay log and that on server are not the same, the update is aborted and the user has to perform manual update.

Support for weak connectivity

- Weak connectivity
 - Intermittence, low bandwidth, high latency, high expense
- Changes
 - Changes in Coda's RPC to function efficiently over a serial line IP (SLIP)
 - Rapid Cache validation
 - * Multiple levels of granularity at which cache coherence is maintained:
 - Volume and object
 - Objects and volumes have version stamps
 - Allowing logical disconnected operation while physically connected
 - * Log updates but still service cache misses
 - * Reintegration initiated periodically by the user

Support for weak connectivity (2)

- Eliminate manual triggering of reintegration – **trickle reintegration**
 - * Updates are propagated to the servers in the background
 - * New state
 - Hoarding state – strongly connected
 - Emulating – disconnected
 - **Write Disconnected** – weakly connected
 - » Updates are logged as in disconnected state and sent to the servers via trickle reintegration
 - » Cache misses serviced as when connected but ask for user intervention in some cases
 - * Replay log optimizations
 - Removing log entries when they become obsolete
 - Aging method used to keep records in the log long enough to get the chance of being removed by the optimization process
 - * Reducing the impact of reintegration
 - Reintegration is done in chunks depending on the current bandwidth

Support for weak connectivity (3)

- Improve handling of cache misses
 - * “Patience threshold” = the maximum amount of time a user is willing to wait for a particular file
 - * User intervention required when the estimated service time exceeds the “patience threshold”
 - Fetch the file or not ?
 - * All misses that can be serviced faster than the “patience threshold” are serviced in the background.

Evaluation

- Rapid cache validation
 - Do volume callbacks help ?

Client	Missing Stamp	Validation Attempts	Fraction Successful	Objs per Success
caractacus	2%	650	97%	40
deidamia	2%	2257	98%	112
finlandia	13%	541	99%	32
gloriana	2%	1457	97%	29
guntram	0%	2977	99%	26
nabucco	1%	1301	96%	28
prometheus	6%	1617	97%	74
serse	8%	1790	98%	32
tosca	1%	652	99%	60
valkyrie	4%	759	96%	32
Mean	4%	1400	98%	47

(b) Laptops

Client	Missing Stamp	Validation Attempts	Fraction Successful	Objs per Success
bach	2%	970	99%	89
berlioz	8%	1178	97%	48
brahms	0%	542	99%	5
chopin	4%	1674	97%	102
copland	3%	1387	94%	171
dvorak	2%	5536	98%	75
gershwin	11%	467	95%	32
gs125	0%	897	99%	22
holst	0%	474	99%	29
ives	0%	1532	98%	56
mahler	1%	566	97%	6
messiaen	0%	827	98%	31
mozart	1%	1633	98%	126
varicose	0%	568	98%	32
verdi	6%	2370	98%	64
vivaldi	7%	344	89%	28
Mean	3%	1310	97%	57

(a) Desktops

- YES

These tables present data collected for approximately four weeks in July and August 1995 from 16 desktops and 10 laptops. The first column indicates how often validation could not be attempted because of a missing volume stamp. The last column gives a per-client average of object validations saved by a successful volume validation.

Figure 9: Observed Volume Validation Statistics