Coda

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Coda

- Coda is a large scale distributed file system.
- Provides a high level of resiliency:
  - Tolerates server failures by having replicas.
  - Allows for disconnected operation. A client can temporarily act as a server.
- Efficient and easy to use.
- Location transparent (similar to AFS).

Historical Overview

- Coda arose out of AFS.
- It needed to provide more fault tolerance.
- Aim: Constant Data Availability
  - Provide data availability in spite of failures in the system.
- Was meant to integrate portable computers in the file system network (read laptops).
- Need for compatibility with Unix file semantics.

Outline

Coda Caching

- Observation: Caching is key to the efficient performance of AFS. Better is the cache, better is the performance
- Clients cache entire files in their disks.
- Uses the AFS caching mechanism as a baseline
    - Check the cache on a file open() call.
    - If the file is not there, fetch it from the server.
    - If the file has been modified, then write it back to the server after the close() call.

Outline

Coda Semantics - I

- *One-Copy Unix* Semantics: Modification to any byte in a file is immediately and permanently visible to every client.
- AFS-I Semantics: Propagate changes at the granularity of files (at the time of open and close only).
- AFS-II Semantics:
  - The client sets up a callback mechanism with the server.
  - It informs the server about its cached files.
  - Whenever a file changes, the server notifies the client.
  - If there is a network partition, the client cache is incoherent .

Coda Semantics - II

- Coda uses a set of servers S.
- A client maintains a subset of servers $s \subseteq S$ that are reachable.
- Every τ seconds, a client recomputes s.
- On an open()
  - A client gets the latest version of a file from s.
  - If $s = \phi$, then it uses its cached version.
- On a close()
  - A client propagates the update to all of s.

Outline

Coda Replication

- Unit of Replication : A volume (a set of files and directories, subtree of the shared file system)
- Each file or directory has an unique ID
- A part of this ID identifies the parent volume.
- A set of servers with replicas of a volume, are known as the volume storage group (VSG)
- The list of servers are stored in the *volume replication database*.
- The client cache manager (*Venus*) keeps track of the subset of the VSG that is accessible (AVSG).

Replication Strategy

- Upon a cache miss, a client obtains the file from one member of the AVSG. (Preferred Server)
- The preferred server can be chosen on the basis of physical proximity.
- The client contacts the other servers on the AVSG to verify the preferred server has the latest copy of the data.
- If the preferred server is outdated, then the server with the latest copy is made the preferred server.
- Establish a callback with the preferred server.
- Upon a file close – it is transferred to all the members of the AVSG.

Cache Coherence

- The client needs to recognize the events not more than τ seconds later.
  - Enlargement of the AVSG.
    - Contact missing members every τ seconds.
    - If an AVSG expands, then cached files may be out of date. Coda drops the callbacks on these files.
    - The next time that these files are requested, the new AVSG needs to be contacted.
  - Shrinkage of the AVSG.
    - Detected by probing each member every τ seconds.
    - If the preferred server dies, then Venus removes its callbacks.
  - Loss of a callback event.
    - Upon a read, the client verifies the version of the file in the preferred server with that of other servers in the AVSG.
    - If there is a mismatch, then there might be a dropped call back.
    - Uses a summary of updates on a volume (volume version vector) as a basis of comparison.

Outline

Efficient Parallel Communication

- Each remote operation typically requires to contact multiple servers.
- Coda provides multiRPC for this purpose.
- MultiRPC uses the multicast capabilities of the network.

Disconnected Operation

- Disconnected Operation begins when the AVSG is empty.
- If there is a cache miss in disconnected mode there is a problem .
- Venus tries to minimize cache misses by using the LRU replacement policy.
  - Coda also allows the user to specify a priority for files.
  - High priority files are not removed from the cache.
- Allows the user to annotate a sequence of actions.
  - Every file generated as a result of those actions is denoted as sticky .

Reintegration

- Happens after disconnected mode ends (one of the servers in the AVSG is up).
- For each modified file, updates are propagated to the servers in the AVSG.
- Proceeds top-down from the leaves.
- There might be conflicts.
  - Provide a temporary home for storing the client updates (co-volume).
  - Similar to lost+found directory in Unix, and vector clocks in dynamo.
  - Let the client resolve the updates later.

Voluntary Disconnection

- When a user voluntary disconnects her laptop.
  - She relies on the large file cache.
  - She needs to re-synchronize later.

Outline

Conflict Resolution

- When a conflict is detected, Coda tries to resolve it automatically.
    - Easy to automatically resolve conflicts on directories.
    - There are three kinds of conflicts that cannot be automatically resolved.
        - update/update conflict: The status of the same object is updated differently in different partitions.
        - remove/update conflict: Updating an object in one partition, and removing it in the other.
        - name/name conflict: Two files with same name are created.
    - Coda has a specialized repair tools that allows the user to fix these conflicts.
        - The user can see all the replicas.

Replica Management

- Each modification has an unique storeid .
- The server maintains a history of storeids.
- If the history of storeids on server A is a subset of that in server B, then B contains newer copies.
  - Coda will consider B to have the latest version.
- This method is useful for files, but can be very conservative for directories.
- Coda maintains the following information:
  - Coda maintains the LSID (latest storage id), and the current length of the update history.
  - LSID $\rightarrow$ client:$<$monotonically increasing integer $>$
  - A replication site also contains the length of the update history of every other replica.
  - $CVV \rightarrow$ A vector containing all the length estimates constitutes

Comparison of Replicas

- Strong Equality : $LSID_A = LSID_B$ and $CVV_A = CVV_B$
- Weak Equality : $LSID_A = LSID_B$ and $CVV_A \neq CVV_B$
- dominance : $LSID_A \neq LSID_B$ and $\forall i, CVV_A[i] \geq CVV_B[i]$
- inconsistency : If none of the other three conditions hold.
- If there is strong and weak equality, the replicas are synchronized.
- If replica A is dominating replica B, then replica B needs to be dropped.

Outline

State Transformation – Update

- Update :
  - Most common operation – file create, delete, modification of permissions
  - First Phase:
    - The client sends the LSID and CVV to each AVSG server.
    - If there are no conflicts, the server performs the desired action.
  - Second Phase:
    - Each AVSG site records the clients view of which AVSG sites performed the update successfully.

Check at an AVSG Server

- The check succeeds for files if:
  - The cached and server copies are the same.
  - Or, the cached copy dominates.
- The check succeeds for directories if:
  - When the two copies are equal
- If the check does not succeed:
  - The client pauses the operation, and invokes the resolution subsystem.
  - If the resolution subsystem can automatically fix the problem, then the client restarts.
  - Otherwise, an error is returned to the client and the operation at the server is aborted.
  - If the operation is successful, the server performs the action, notes the LSID of the client, and commits a temporary CVV.

Update Operation

- At the end of phase I, the client examines the replies from each server.
- For each responding server *i*, it augments $CVV[i]$.
  - The client sends this CVV to every responding server.
  - Each responding server replaces its tentative CVV by this CVV.
- Venus returns control to the user at the end of the first phase.

State Transformation – Force

- Force operation – Transfer of file contents from a dominant to a submissive site.
- Force of a directory is more complex.
    - Lock and atomically apply changes one directory at a time.
    - Before creating a new entry, we first create a stub at the server. It contains a CVV that will always make it submissive.
    - Subsequently, a force operation will change the status of the stub.

State Transformation – Repair and Migrate

- A repair operation is used to fix inconsistent updates.
- If we detect inconsistent updates, then the file is marked as inconsistent and moved to a covolume .
- All accesses to inconsistent objects fail.

Implementation

- Implemented on IBM workstations.
  - 12 MB main memory, 70 MB Hard Disks
- Each server had 400 MB disks
- Uses the Camelot transaction facility for single site transactions.
- Uses the Andrew file system benchmark
  - 70 files – 200 KB each

Cost with Replication

Configuration	Time Overhead
No Replication	21%
1 Extra Server	22%
2 Extra Servers	26%
3 Extra Servers	27%

Benchmark Time vs Load

- For AFS the elapsed time remains roughly constant at 400 seconds (1 to 10 load units).
- For Coda the time increases from 400s to 650s roughly quadratically for 1 to 10 load units.

Benchmark Time vs Load

- Iterative Unicast : The network load in terms of packets increases linearly from 5,000 to 60,000 while varying the load units from 1 to 10.
- Multicast : For the same range of load units the network load increases linearly from 5,000 to 40,000.

Coda: A Highly Available File System for a Distributed Workstation Environment by Mahadev Satyanarayanan, James Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere, IEEE Transactions on Computers, 1990