**CODA SUMMARY**

CODA is a file-system which provides resiliency to server and network failures and is mainly used in large-scale Unix based distributed systems. Resiliency is provided by two basic but very effective mechanism, one is server replication which stores a copy at multiple servers and the other is disconnected operation where caching site temporarily acts as a replication site and it supports portable workstations. The design is optimized in such a way that high degree of consistency is achieved in availability and performance. Coda File system is a descendent of AFS(Andrew File System) which was very successful among distributed workstation environment but people were concerned with the vulnerability to failure of server’s and network component’s and hence coda was introduced which was mainly focused on improving this vulnerability. Coda strives for consistent data availability where incase of failure user’s are allowed to use local resources instead of shared repository to continue the work-flow. One of the important task of coda is to integrate AFS to portable computers so that user’s can copy some data, work on it in their portable device and replace it with the new one. If the connection is not stable it also results in fault tolerance. Three fundamental factors that influence the design of coda are: desire to produce highly scalable system, working on failures and minimizing them, need to emulate Unix file semantics. Due to the growing economy it is very important for a system to be scalable as the number of users increase day-by-day and also its performance should be maintained with minimal cost. Also in AFS caching is an important mechanism which is used improve the scalability where most of the load is carried by the client. Whereas in Coda clients dynamically maps the information to server and it uses token-based authentication and end-to-end encryption. Next is to minimize the range of failure there were two strategies, first one was replication across servers and other one is to deliberately make capable of full autonomous operations. But both of these operations fail. Hence a combination of two operations were built to reduce failure where coda uses service replication where the copy of server is stored in multiple systems and if the servers fail the clients gets disconnected and uses the cached data for processing information. Also these operations can be performed alone. One-copy Unix semantics is required to make coda failure proof and shared Unix file system. To do so, accuracy has been reduced and approximation is achieved. To improve the approximation two lessons were taken from AFS. They are: AFS-1 Semantics where each time a file was opened and closed changes were made and stored separately and later if the file opens it specifies that this is the latest copy. The second lesson is AFS-2 Semantics where callback is produced even for the weakest system and file fetching and validation takes place. In Coda semantics a single server is replaced by multiple servers and gets updated every ‘t’ seconds. When user asks for information and no server is available, data is taken from the cached data. Callback and optimistic replication scheme weakens currency guarantees. Also Coda’s currency guarantees is said to weaker for larger applications. Hence coda does not support database since huge data’s will be stored. Server replication is an important part of coda where each unit is represented as volumes and are stored in volume replication database at every server. The strategy used for replication is read-only write-all where client obtains data only from the preferred server which is chose based on the performance. Also if the client is connected to anyone of the server’s it can access data from all the servers. Also these server’s avoid conflicts since they are optimistic. For cache coherence in coda, client needs to recognize three kinds of events where enlargement of volume group so that missing members can be added every t seconds. Shrinking of volume group is detected if normal operation of volume group fails. Also shrinking will help us to detect lost callback. Only the preferred server maintains a callback from which all other servers are notified. Volume version vector is introduced to notify other clients connected to their respective server so that they can notify a callback. Also Coda uses parallel communication since server replication Is performed multiple sites are connected. MultiRPC is used for parallel communication since it is optimized and helps in faster communication. When no member of VSG is accessible disconnected operation occurs and client moves to the previous state in the least time. Cache miss is transparent in normal operation whereas in a disconnected operation it is seen and it is very important and it continues till it is reverted back to normal operation or it is aborted. Reintegration process starts when the disconnected operation ends. It helps to integrate all the modified and deleted files during disconnected operation. Reintegration can fail in two cases, when there is no authentication tokens and inconsistencies maybe detected due to updates. Disconnected operation can also occur voluntarily when a client is deliberately disconnected from the network. When a conflict is detected coda performs automated resolution where it changes the name and id and tries to resolve the conflict by itself. Else it performs repair tool operation where it allows the user to manually solve the problem. Replication management helps us to know about the services and protocols used for server replication. Each modification the server has a storeid using which the operation performed can be found out and the characterization can be computed using which similar replicas can be found. Coda maintains an approximation of current length of update and its storeid. State representation helps us to find out the current state of the updated values and similar values using the storeid. There are four operations in state transformation, they are: update, force, repair, migrate. Update is the most commonly used class in mutation. Update is used almost in every operation when we get a new value. In force operation the client forces itself in server-server operation just to transfer the data and to ensure smooth process. Repair and migrate are rare operations where repair is used to fix incase of any errors. Migrate operation is used in disconnected operation to place an object at the end. Performance evaluation is used to analyze the process of coda in different perspective. Server replication is mainly used during failure so that the process should continue even during failure. Server replication affects performance degradation since it uses complex protocols. When multiple workstations are added it takes a long time to process it than AFS does. So coda does not balance load in a proper way. Multicast helps in balancing the load and it produces better results than operation without multicast. It is useful in two ways, reduces the latency of storing large files and network load. Ramdisk is used to reduce the latency of update operation. Ramdisk copies data in shorter time compared to other devices. Thus providing sustainability and availability are the main operations of coda and it has been achieved using two mechanism, server replication and disconnected operation. The only drawback is the loss of performance which should be corrected in the upcoming updates so that it becomes a complete powerful system with high availability.