Build a highly available, replicated DNS server that uses Viewstamped Replication

Students Aasneh Prasad (210101001)
Pranav Jain (210101078)
Sanjana Siri Kolisetty (210101093)
Professor Diganta Goswami, CS542 - Distributed Systems
Institute IIT Guwahati, India

Problem Definition

Build a highly available, replicated DNS server that uses Viewstamped Replication to ensure consistency of updates.

VR differs from Paxos in that it is a replication protocol rather than a consensus protocol. It uses consensus, with a protocol very similar to Paxos, as part of supporting a replicated state machine.

Survey of related literature

PAXOS

Paxos, introduced by Leslie Lamport in the late 1970s, is a consensus algorithm designed for asynchronous networks where nodes can fail. Its primary goal is to ensure that a group of nodes can agree on a single value even if some nodes fail or messages are lost. The Paxos protocol consists of three roles: proposers, acceptors, and learners. Proposers suggest values, acceptors vote on these values, and learners learn the chosen value once a majority of acceptors have agreed. Key Characteristics of Paxos are:

- **Consensus Focus:** Paxos is fundamentally a consensus protocol, meaning it is designed to reach agreement among distributed nodes.
- **Fault Tolerance:** It can tolerate up to f failures in a system of size $2f + 1$.
- **Disk I/O Requirement:** The original Paxos implementation often requires disk operations to ensure persistence and recoverability.

Viewstamped Replication

Viewstamped Replication was developed around the same time as Paxos but primarily focuses on state machine replication rather than consensus alone. It provides a mechanism for replicating services across multiple nodes while ensuring that all replicas maintain the same state. Key Characteristics of Viewstamped Replication are:

- **Replication Protocol:** VR is designed to replicate state machines, allowing for operations to be executed consistently across replicas.
- **View Changes:** VR employs a view mechanism where one replica acts as primary in each view, facilitating request ordering and handling failures through view changes.
- **No Disk I/O Requirement:** Unlike Paxos, VR does not require disk I/O during consensus operations, which simplifies its implementation and potentially improves performance.

They address the challenges of consistency and fault tolerance but cater to different needs within that spectrum—Paxos for consensus and VR for service replication.

- The *configuration*. This is a sorted array containing the IP addresses of each of the $2f + 1$ replicas.
- The *replica number*. This is the index into the configuration where this replica's IP address is stored.
- The current *view-number*, initially 0.
- The current *status*, either *normal*, *view-change*, or *recovering*.
- The *op-number* assigned to the most recently received request, initially 0.
- The *log*. This is an array containing *op-number* entries. The entries contain the requests that have been received so far in their assigned order.
- The *commit-number* is the *op-number* of the most recently committed operation.
- The *client-table*. This records for each client the number of its most recent request, plus, if the request has been executed, the result sent for that request.

Figure 1: VR State at each replica

Overview of Viewstamped Replication

Viewstamped Replication (VR) is a replication protocol designed to ensure consistency and availability of services in the face of node failures. The protocol operates in an asynchronous network and is particularly focused on handling crash failures, where nodes either function correctly or are completely down. VR utilizes a primary-backup architecture where one replica acts as the primary to order client requests, while the other replicas serve as backups. This structure allows the system to manage client operations effectively while ensuring that all replicas maintain a consistent state. The protocol is designed to handle scenarios where replicas may fail or become unreachable.

Assumptions

- **Asynchronous Network:** VR is intended to work in an asynchronous network where the non-arrival of a message indicates nothing about the state of its sender.
- **Message Behavior:** Messages might be:
  - lost,
  - delivered late or out of order, and
  - delivered more than once.
- **Eventual Delivery:** It is assumed that if sent repeatedly, a message will eventually be delivered.
- **Client Request Constraints:** A client is allowed to have just one outstanding request at a time. The client records its own client-id and a current request-number to prevent a request from being processed more than once and to prevent multiple responses.

The core functioning of the VR protocol through three critical sub-protocols

Step 1. Normal Case Processing of User Requests

- **Client Request Handling:** Clients send requests to the primary replica, which checks if the request is valid based on its client table. If valid, it logs the request and sends it to the backups for processing. Each message sent to the client informs it of the current view-number, which allows the client to track the primary replica.

- **Replication Process:** The primary sends a `PREPARE` message to other replicas and waits for acknowledgments (`PREPAREOK` messages) from a quorum of backups before considering the operation committed. Once committed, it executes the operation and sends a reply back to the client.
- **Backup Coordination:** Backups process messages in order and must have all earlier requests logged before accepting new ones. When a backup learns of a commit, it waits until all previous operations are committed before logging the most recent operation. This ensures consistency across all replicas.

Step 2. View Changes to Select a New Primary

- **Failure Detection:** If the primary fails (timeout expires without any `PREPARE` or `COMMIT` message from the primary), backups detect this failure. They use view changes to mask failures of the primary by sending a `STARTVIEWCHANGE` message to other replicas. When a replica receives `STARTVIEWCHANGE` messages for its view-number from f other replicas, it sends a `DOVIEWCHANGE` message to the node that will be the primary in the new view.
- **Ensuring Consistency of Logs:** The view change protocol obtains information from the logs of at least $f+1$ replicas. This is sufficient to ensure that all committed operations will be known, as each must be recorded in at least one of these logs.
- **New Primary Selection:** A view change protocol is initiated where backups elect a new primary from among themselves, or a new primary is elected using a round-robin scheme. This process ensures that operations can continue even if the original primary is down.
- **State Synchronization:** During a view change, it is crucial that the new primary has knowledge of all operations that were executed in previous views to maintain consistency. To achieve this, it sets its log to the latest log obtained in all `DOVIEWCHANGE` messages.

Step 3. Recovery of a Failed Replica

- **Rejoining Protocol:** When a failed replica recovers, it must synchronize its state with the current state of the system before rejoining. While a replica's status is recovering, it does not participate in either the request processing protocol or the view change protocol.
- **State Transfer:** The recovering replica requests missing information from other replicas by sending `RECOVERY` messages to ensure it has an up-to-date view of the system's state. Other replicas reply by sending `RECOVERYRESPONSE` messages.
- **Quorum Requirement:** The recovering replica must be aware of the states of at least $f+1$ replicas to ensure it can participate correctly in future operations. It then updates its state using information from the primary, changes its status to normal, and completes the recovery protocol.

Implementation in GO Language

The GitHub repository (https://github.com/SanKolisetty/Distributed-Systems-Term-Paper) presents an implementation of the Viewstamped Replication (VR) protocol in the Go programming language. The system is structured around a set of replicas that communicate with clients through a proxy layer. Each replica executes the VR protocol, ensuring that all instances maintain a consistent state despite potential failures.

It implements VR in the following steps:

- **Client-Replica Interaction:** Clients send requests to the primary replica, which manages the order of operations and coordinates with backup replicas.
- **Request Processing:** The code implements a robust request processing mechanism where the primary logs client requests and ensures that a quorum of backups acknowledges these requests before they are executed.
- **View Change Mechanism:** Backup replicas can elect a new primary, ensuring continuous operation without requiring complex consensus rounds.

Conclusion

The development of a highly available, replicated DNS server using Viewstamped Replication (VR) demonstrates a robust approach to achieving consistency and fault tolerance in distributed systems. By leveraging the VR protocol, the system ensures that all replicas maintain synchronized states, even in the presence of node failures. VR is specifically tailored for state machine replication rather than general consensus. This focus allows VR to efficiently handle client requests through a primary-backup architecture, facilitating seamless recovery and view changes when necessary.

Overall, this project not only reinforces theoretical concepts surrounding distributed systems but also provides practical insights into building robust architectures capable of sustaining high availability and consistency.

References

[1] Barbara Liskov and James Cowling. *Viewstamped Replication Revisited*. Available at: https://pmg.csail.mit.edu/papers/vr-revisited.pdf

[2] Leslie Lamport. *Paxos Made Simple*. Available at: https://lamport.azurewebsites.net/pubs/paxos-simple.pdf

[3] *Understand Viewstamped Replication with Rust, Automerge, and TLA+*. Available at: *https://medium.com/@polyglot_factotum/understand-viewstamped-replication-with-rust-automerge-and-tla-7a94e9e4d553*