Distributed Key-Value Store

proj2_e9t9a_l8m0b_r9t9a_s1c9
Seerat Sekhon, Shavon Zhang, Winnie Wong, Brendon Wong

Introduction

A distributed key-value store is a key-value store hosted in multiple instances or separate datastores. The purpose is to have better speed, prevent a single point of failure, be able to store more data at lower costs, have incremental scalability, and load-balancing.

Overview

What problem can we solve?

- Consistency: all nodes will return the same data eventually
- Availability: fault-tolerance, network fails
- Scalability: Coordinator node distributes requests among nodes (Load Balancing)

Design

Architecture
Node Topology:

- The nodes in the network are connected in a ring structure
- the ring of nodes will have one coordinator per data-center. It will be elected among servers using some protocol (Paxos?)
- One node will be the request coordinator which will load balance client read/write requests

Behaviour

1) Membership protocol for nodes
   it is necessary to keep all nodes informed at all times of the other nodes in the ring. At each node, the key-value store talks to the server and receives from it the membership list. Periodically, the node polls the membership protocol to bring its membership list up to date. This will be done similar to Project 1. (Circular array)
   How to insert nodes into the ring?
   Authentication of nodes → Public Private key
   Deciding which nodes are replicas for this node (2f + 1)
2) Storage at each node
   Hashmap of Key to Value
3) Replication/Consistency:
   The type of consistency we are trying to achieve is called eventual consistency. Eventual consistency means that if all writes stop, all replicas of each key will eventually have the

same value.
Consistency:

a) Logs: remove commits that aren't needed anymore. Ex: remove put operation only after delete op received for same key
b) Replication
c) Quorum with Coordinator

4) Node Conflict Resolution
   Gossip Protocol

5) Availability (Node Failure)
   This is required in order to handle an eventual node failure. When a node fails, the key->node mapping breaks for some keys. As it fails, there will be n-1 nodes for key distribution, meaning that many keys will suddenly have a different set of nodes assigned. In this situation, stabilization comes to rescue, re-replicating keys whose replica(s) failed. Naive solution is to go through each key in each node, check if the key still belongs to the node (primary, secondary, tertiary) and handle it accordingly.
   a) Heartbeats, timeouts (Gossip Protocol)
   b) Replication
   c) Caching

6) Client Reads
   a) Request to node/coordinator
   b) Coordinator determines primary, secondary, tertiary replicas
   c) Ask replicas and wait for a majority quorum answer

7) Client Writes
   a) Request to coordinator/node
   b) Coordinator determines primary, secondary, tertiary replicas
   c) Ask replicas and wait for a majority quorum answer

Testing

Unit tests
Manual testing
Load testing (?)
Azure

SWOT Analysis

Internal
Strengths

- Team members are familiar with each other

- Team members are highly available for communication
- Team members have a diverse background with various tools and technologies
- Familiarity with coding in Golang

Weaknesses
- Writing tests in Golang
- Coordinating with in-progress components on a self-designed project
- Team has had issues meeting deadlines

External
Opportunities
- Many tutorials and resources available for creating KV stores

Threats
- Assignments, exams and other commitments can hinder the completion of this system
- Abundance of resources may cause confusions in design and implementation

References

1. https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
2. http://blog.fourthbit.com/2015/04/12/building-a-distributed-fault-tolerant-key-value-store
3. https://www.quora.com/What-are-good-ways-to-design-a-distributed-key-value-store
4. http://jodah.net/create-a-distributed-datastore-in-10-minutes
5. https://www.allthingsdistributed.com/2007/10/amazons_dynamo.html