CS 425/ECE 428
Distributed System

MP2 Key-Value Store

Design Documents

Group Member:

Name: Rui Xia
NetID: ruixia2
UIN:660935771

Name: Youjie Li
NetID: li238
UIN: 668129373

Spring 2017
April 10, 2017

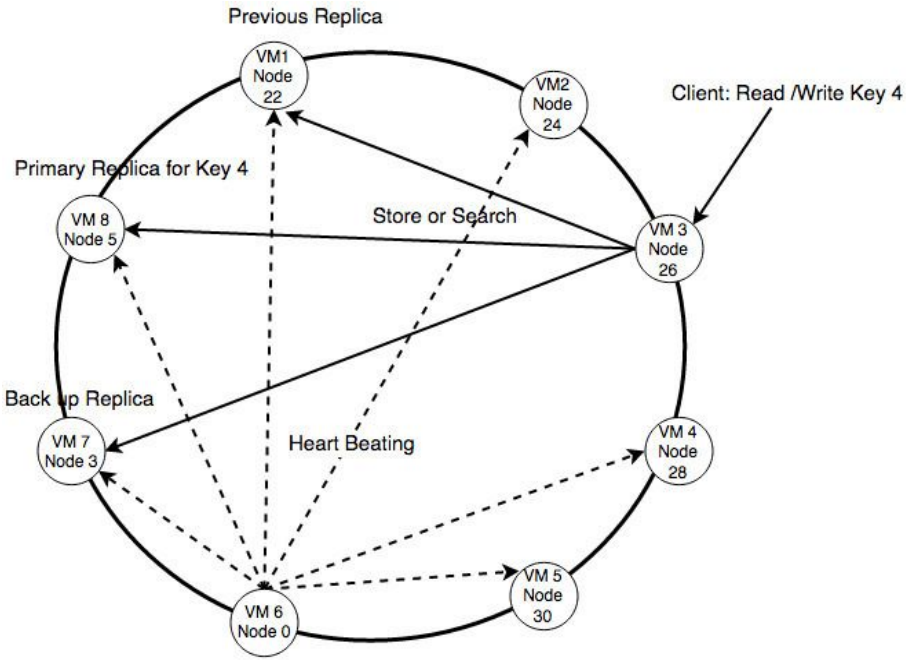

Figure 1. Structure of DHT

Algorithm Selected

For this MP, we choose Cassandra, the Ring-Based DHT without finger table or routing key, to implement the distributed key-value store on 10 VMs. The reason of our choice is that the Cassandra is scalable. To be specific, each node knows the information of each other so that the time complexity for search operations will be just O(1). Also, servers can directly hash the data and find the right node to store. We set the hashing bits as 5 and mapped 10 node id from 0 to 31. The structure of the ring-based DHT is shown in Figure 1. All-to-all heartbeats are sent periodically for failure detection and new join detection.

Impelimentation Details:

The general flow chart is shown in figure 2.

A. Commands

- **SET**
  Hashing key id directly and find three nodes in which the replicas should be stored (primary, backup, and previous replicas). Then “store” msgs are sent to those nodes. Server who received a “store” msg should store the data into its local memory.
- **GET**
  Similar to SET, hashing keys and finding replicas. Then asking for the corresponding nodes with“search” msg. Nodes received the “search” will search the key locally and send back the value with a timestamp. The GET method will return the latest value sent back among those three replicas, in order to achieve consistency.
- **LOCAL_LIST**
  Each node prints out all data in their local memory.

- **OWNERS**
  Similar to GET, finding replicas and asking for response. All nodes who contain the key will sent back their IDs such that the owners of the key are posted.

B. Recovery

- When failures happen, the predecessors and successors will copy replicas from their local memory to pre-predecessors and suc-successors in order to keep three replicas in total. The copying operations will be implemented as sending "store" msgs.
- In order to deal with two simultaneous failures, we serialize the two recovery processes. That is to block the second recovery until the first recovery completes.

C. Rebalance

- A new join node will be detected as long as its first heartbeat is received.
- Similar to recovery, if new join is detected, the successors should copy parts of its local memory to the new nodes. At the same time, predecessors and suc-successors should delete the corresponding data from their local memory in order to keep exactly three replicas in total over the entire system.
- To deal with two simultaneous joins, we also serialize the join processes. That is to block the second joining until the first completes. A boolean flag, "rebalancing", is utilized to implement the serial rebalancing.
- Moreover, the rebalancing flag are also set as true during the recovery such that a new rebalance will be delayed until current recovery completes.

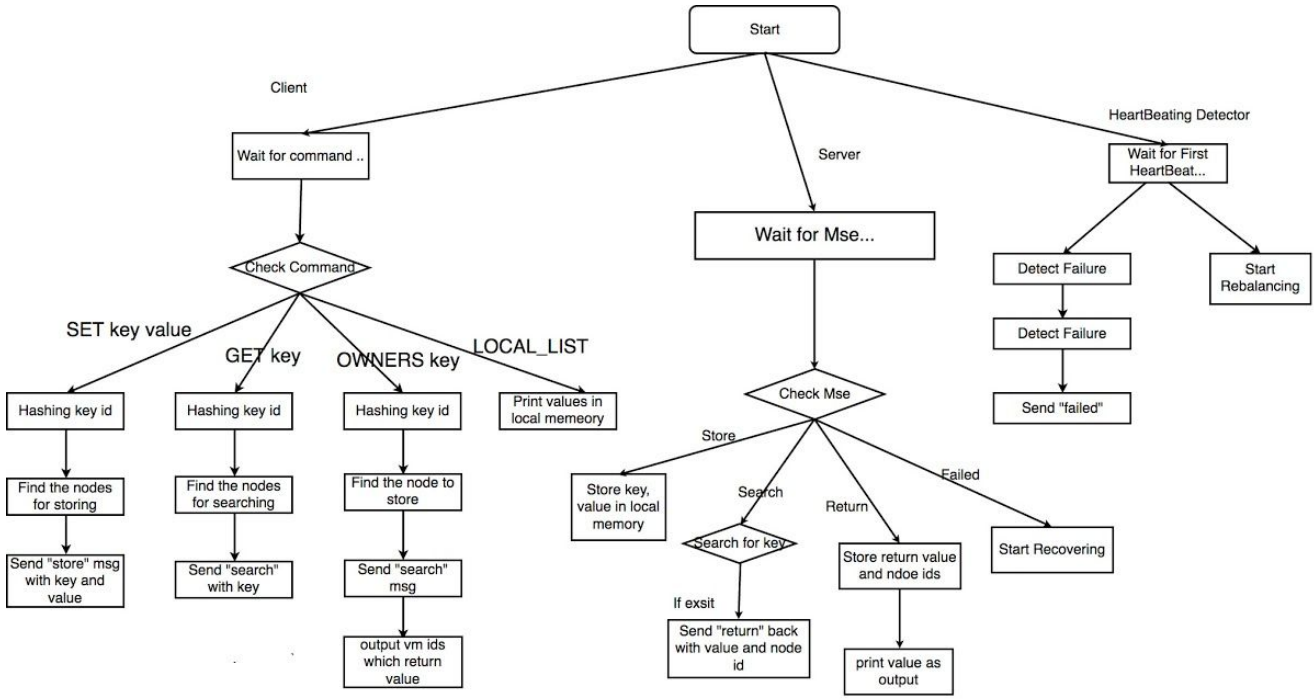

Figure 2. Flow Chart for Algorithm Design

Implementation Parameters:

- Period of Heart Beating: T= 3 s
- Timeout for one or two failures: T + delta <= 6s (T=3s)
- Hashing bits: m = 5
- Number of replicas: k = 3

Theoretical Estimation for the N-node System

- capacity = number of hashed IDs can be stored = 2^m (> N)
- memory usage per node = O(N)
- bandwidth usage per lookup = 2k (<< N)
- latency per lookup = one message RTT
- failure detection bandwidth usage = O(N) per node
- failure detection time = T + delta (delta relates to max one-way delay)

Evaluation

We have implemented our key-value system using the parameters mentioned above. In order to test the scalability and performance of our system, three major metrics are evaluated, which are the worst-case failure detection time, failure detection bandwidth per node, latency per lookup. Firstly, we have tested multiple independent failures in the system and measured the worst failure detection time. Secondly, we have evaluated the heartbeat bandwidth usage of each node over 10 mins. Thirdly, the latencies of the search operation which returns the values from all replicas are measured as well.

# of Nodes	Worst-Case Failure Detection Time (ms)		Failure Detection Bandwidth per Node (# of msg / second)		Latency per Lookup (ms)	
	Average	95% Confidence Interval	Average	95% Confidence Interval	Average	95% Confidence Interval
4	6252.732913	(6102.732913 to 6402.732913)	0.9995831791	(0.9994431791 to 0.9997231791)	248.673738	(246.9673738 to 249.9673738)
5	6264.083982	(6094.083982 to 6434.083982)	1.332691237	(1.332181237 to 1.333201237)	226.0504484	(209.0504484 to 243.0504484)
6	6303.951931	(6143.951931 to 6463.951931)	1.700566257	(1.677566257 to 1.723566257)	249.8318354	(218.8318354 to 280.8318354)
7	6297.753255	(6127.753255 to 6467.753255)	2.005982028	(2.000882028 to 2.011082028)	347.5894451	(291.5894451 to 403.5894451)
8	6536.640338	(6356.640338 to 6716.640338)	2.332193098	(2.3318230979 to 2.332563098)	293.282485	(254.282485 to 332.282485)
9	6320.541819	(6170.541819 to 6470.541819)	2.665224527	(2.664554527 to 2.665894527)	274.186945	(240.186945 to 308.186945)
10	6521.601701	(6452.601701 to 6590.601701)	2.954707147	(2.950607147 to 2.958807147)	267.3580117	(246.3580117 to 288.3580117)

As for the worst-case failure detection time, the theoratical value is (T + 2*max-one-way-delay) while the practical value is (Timeout + max-one-way-delay + program overhead) which is around 6300 ms. As can be seen, this factor is scalable with an increasing number of nodes.

As for the failure detection bandwidth, it increases with the size of the system, which is as expected, because all-to-all heartbeat is adopted in this design in order to support arbitrary number of simultaneous failure and join.

As for the latency per lookup, it depends only on the RTT and program overhead. As can be seen, this factor is also scalable.