CSC469: Assignment 3

Daniel Murphy
murphyd8 999825033

December 11, 2016

1 Introduction

A good system of distributed key-values needs to provide quick and scalable service to servers which can store and retrieve memory, rather than traditional SQL databases. This distribution of keys across multiple servers allow greater resilience in the face of faults, as servers who have faulted will have adjacent servers how can restore and handle requests while a server is recovering.

2 Analysis

Our resign begins with the a single metadata server(or mserver) M and n many servers which store the key-value pairings. Request are sent in by clients to be handled first by a metadata server then sent across to a corresponding server. Each server stores the key-values in a hash table, which store it?s key-value pairing and an additional hash-table storing a duplicate replica for an adjacent(the server?s secondary) server.

Should a server, named S_a, fault and become inoperable, the metadata server will detect that server S_a has gone down and begin a recovery protocol. The metadata will spawn a new server S_{aa} to replace the faulted server and initialize data for the recovering server. Adjacent servers S_b(which stores the replica key-value table in its secondary duplicate table) and S_c(which is the primary key-server table which S_a stored in its replica key-values) will send their secondary and primary replicas to server S_{aa} ?s primary and secondary hash-tables.

For failure detection, servers will send periodic heartbeat messages to the metadata server, such that if a server faults the metadata server will know that the server has faulted because it has not sent any messages to it.

3 Final Analysis

We were able to implement a majority of the features outlined in the paper, but were unfortunately unable to implement all of the features outlined in the assignment handout. Failure detection was successfully implemented such that when a server faults and no longer sends heartbeat messages, the metadata server which receives those messages will know that the server is down and will start recovery. Recovery has also been implemented, such that Server S_b and S_c will know to send asynchronously their key-value's to the recovering server such that the faulted server will recover their primary key-value set and replica(or secondary) set as well.

Client Redirection was partially implemented, such that the Server S_b, which is sending its replica to server S_{aa}, which will handle client requests intended for the recovering server then send the client requests in the correct order such that there are no race conditions with the asynchronoues recovery, so stale values won?t replace the newly handled values. The only part we were unable to implement was server S_b unable able to flush all in-flight requests to server S_{aa} after the recovering server S_{aa} had all of its previous key-values restored in it primary and secondary hash-table.

4 Follow Up Questions

Q1. What happens if a server could run out of storage space for its keys? Rejecting client requests due to being out-of-memory is not a practical way to handle this case (as the starter code does). How would a real-world key-value store handle such a case? Explain your design decision, potential alternatives, and discuss their merits or drawbacks.

If a server using our implementation runs out of memory for it?s key, then the system will return

an "Out of Memory" error message, and will reject the client request to add any additional keys. This is impractical as stated in the above/prompting question, and the way I believe a real world system would handle this case by creating or setting one(or more) server(s) to be a wildcard server(s) which handle any requests not able to handle any more keys.

This implementation would additional require additional structuring and functions as the mapping of keys would become partially redundant as well as potentially expensive performance-wise to look into a lookaside/wildcard server every time a key does not exist in the corresponding server it is mapped to.

This better serves as a better system then putting a key in an adjacent server should the one it is mapped to is at max capacity, for all of the functions for mapping keys to servers would become invalid or redundant.

Q2: What happens if the items in a primary replica were striped across several servers, instead of duplicated on one server only? Explain how this would affect your design, in terms of performance, consistency, resiliance in the face of failures, and failure recovery approach.

In terms of how the system would be designed, if server A was to go down, then servers B, C, D, and onwards until N that will all have to sent their replicas to server A, thus the Metadata server would have to send and receive update messages from those servers. In addition, all of the other servers, i.e. the ones from B to N, all have to send their back up keys to server A, as well as likely locking and unlocking needed to protect the key-value stored on each server. I would view this implementation worse performance wise due to the need to go through as many necessary servers which contain the replica keys needed to restore the faulted server?s key-value set, in addition restoring its own replica keys needed to restore other servers, should they fault.

This new design I believe would have the same ability to provide an acceptable level of service in the face of failures as a network with only one replica, as the user which needs to access some information needed that is located on a recovering faulted server will still be redirected to the server holding the replica. Both serve the same end mission of restoring a defaulted server, but I believe the consistency and performance of a duplicated replica on adjacent servers rather than replica data striped across multiple servers, due to the single replica server only needing one mutex lock for the asynchronous restoring another for handling clients, rather than n many for the n servers the data is replicated across.

Q3. What happens if failures could happen during recovery? Explain your rationale.

There are multiple outcomes to the question of what should happen when a failure occurs during recovery. If the server which faulted originially fails agian, then the metadata should restart the process of recovery again, as if starting back at step 1 in the handout. If it happens to any other server, then there are two possible outcomes; first, if it happens to an adjacent server which is helping recover the faulted server, then it will impossible to restore the recovering server(as potentially all of it?s data has not been transferred) and the newly recovering/formerly faulted server(as the recovering node can have none of to all of it?s previous information)

Q4: How many replicas would we need to tolerate N failures? Explain your rationale.

For worst case asssumption, we will need f replicas for f failures, as for every failure, we will need to restore that server?s data. Thus if we have n+1 servers we can have n failed servers, as the lone server will have all of the replicas plus it?s own set.

Q5: This key-value store does replication on the critical path (PUT requests are not acknowledged to the client until the update propagates to the secondary replica(s)). If you were to change this to an eventual consistency model, how would that work? How would you design the recovery mechanism, if replication was not done on the critical path?

I believe it might be feasable if the metadata server does not redirect client requests from the recovering server. Instead any server messages attempting to restore the key-values in server S_{aa} coming from server S_b will be compared and if a key already exists in the newly recovered server S_{aa} then the message from S_b is considered stale and discarded.

Q6: If nodes can be added dynamically in response to increasing load, or if nodes can be taken down when underloaded, what implications would this have on the regular operation and recovery?

To ensure that the servers have correct data and file descriptors for their primary and secondary servers, the metadata server will have to send message to all of the servers when a node server is be-

ing loaded/unloaded. Servers would have to ensure their file descriptors up to date to ensure correct piping. Servers which also have adjacent servers that go down or spawned will have to flush their entire replica array, where we might have two servers that are in recovery, not because the went down but because their data is now obsolete.

For recovery and regular operations, this could take potentially longer times as there is no efficent way to map keys to servers. This is however could not be that big of a factor as Amazon?s Dynamo storage system, which uses many techinques explained above, and when server nodes are loaded and unload, the only n/k keys need to be remapped.(n being the number of server nodes and k being the number of keys) What would be important is that all keys are not lost or dereferenced in this period of remapping key-values should a fault occur, or recovery would be useless.[1]

5 References

[1] G. DeCandia, D. Hastorun et al., Symposium on Operating Systems Principles 'Dynamo: Amazon's Highly Available Key-value Store' in SOSP, 2007.