Shivam Bharuka (bharuka2)
Ahmad Syafiq bin Shamsul Bahari (shamsul2)

ECE 428 Distributed Systems
MP2 Design Document

Key-Value Storage Implementation

We have implemented our algorithm based on Chord and Cassandra, which has eventual consistency. We use the concept of virtual nodes that are arranged in a ring-based network and their locations are predetermined based on the maximum number of nodes allowed to join. When we are allowing more nodes to join, we can simply increase the maximum number of virtual nodes for scalability. We are also using TCP protocol for the connections between the nodes so that messages are always eventually delivered and is used for failure detection as well.

When a node joins a network, it has a network ID and joins a virtual node based on the ID. Each node contains a list of alive nodes in the network and a dictionary that maps the hashed value of a key to the node responsible for storing it. Using the dictionary, we will have constant lookup time complexity when finding the owners (node, successor and predecessor) of a particular key, because everyone will update the dictionary whenever a node joins or leaves the network. The algorithm also ensures that a key stored in a node will have two replicas which is stored in the predecessor and successor of the node. Eventually, there will be three copies of a particular key stored in three different nodes to tolerate up to at most two simultaneous failures. For scalability, we can increase the number of replicas when the number of simultaneous failures tolerable increases.

When a node receives the command SET, GET, or OWNERS, it will hash the key to find the node responsible for it from the dictionary and sends a message with the parameters type which is the command, the key, and value to the corresponding node and sets a timeout afterwards. For GET, the value will be set to ‘’ by default and for OWNERS, both the key and value will be set to ‘’ by default. If the node detects that the receiving node has failed while waiting for a response, it will update its list of alive nodes in the network and the dictionary, and sends the message again to the new node that is responsible for the key based on the new dictionary, before resetting the timeout. If the sender times out before getting a reply message, it will check the dictionary and retry, sending the message again. Otherwise, it prints the string from the response message and now can accept more commands.

When a node receives a message, it will first check the type parameter of the message and calls different functions accordingly. For the type SET, the node will find its successor and predecessor from the dictionary before sending the same message to the two nodes to update the replicas. The node will then add the key if it does not exist in its data or overwrite it if it already

exists. Afterwards, the node will send a reply message that contains the string “SET OK”. For the type GET, it will check if the key exists in its data and sends a reply message with either the string “Found: value” or “Not found”. For the type OWNERS, the node will check its successor and predecessor and responds with a message that contains the string with all three IDs (the receiving node, successor & predecessor) separated with a space.

For the command LIST_LOCAL, no messages are sent, and the node just goes through its data and prints out the key in a sorted order, separated with a newline character and prints “END LIST” at the end. For the command BATCH, it will look at the first file and extract the information line by line before running them as commands one at a time. When it receives the string from the response message, instead of printing them into stdout, the strings will be saved into the second file, with a newline at the end of each line. Each node can only run 1 command at a time, and this ensures that after the execution of BATCH, the output strings in the second file will always be placed on the same line as the input/first file and in the same order as well.

Whenever a node fails, we enter recovery period and all other nodes will be able to detect the failure and update their list of alive nodes and dictionary. The only nodes that will be leading the work during this recovery period is the successor of the failed node, which is known by everyone through the dictionary. It will merge the failed node’s replica into its own data and sends the new data to its successor to update its replica. It will then sends its data to its new predecessor to be saved as a replica and at the same time also asks its new predecessor for a replica of its data.

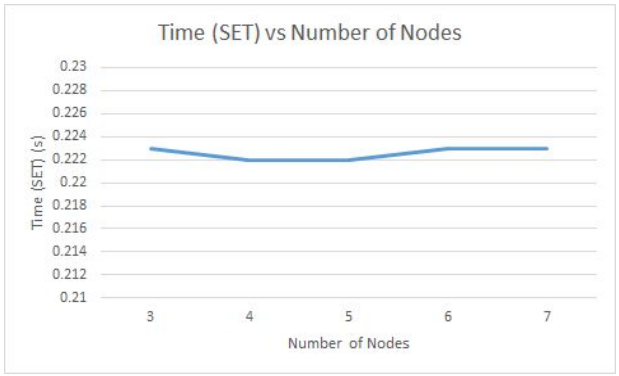
If a node joins, the network, all other nodes will also update their list and dictionary, while the new node will communicate with its new successor and predecessor to update its keys and replicas. Its successor will check the keys which belong to the new node and sends those. It will eventually remove them from its own data after it has gotten acknowledgement that the everyone has finished updating their datas and replicas.

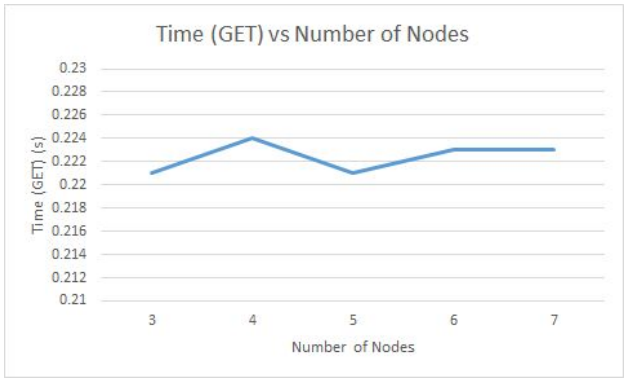
We chose this algorithm over RAFT because RAFT is a lot harder to implement and we will have to wait sometimes to get the result when sending commands. This would happen when an election is being held because there is no leader responsible for handling requests at that time. We have also decided to use the value given by the main node responsible for a given key, instead of getting responses from a quorum or all the nodes that have the key. This is mainly to increase our speed and at the same time, eventually all the key values in the three nodes will updated and be the same.

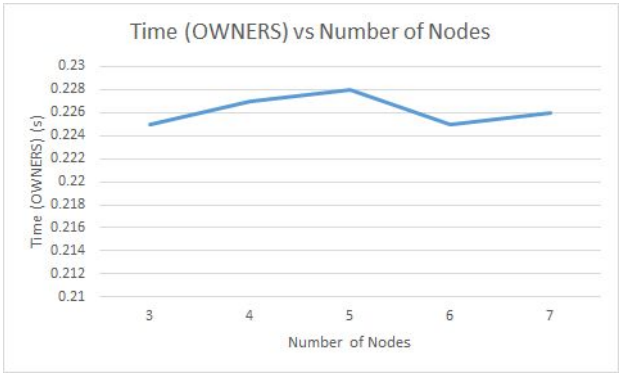
For this algorithm, with the increasing number of nodes in the cluster, the capacity/memory usage of the node will be less because the keys will be approximately evenly

distributed across the nodes, balancing the loads among the nodes with the stabilization protocol. Bandwidth usage per lookup will always stay the same, which is two messages worst case and 0 messages best case. This is because the node only needs to send a message to the node responsible for the key and will get a reply with all the nodes responsible for the keys. If the node is itself, it does not need to send any messages. The latency will also always remain constant because lookup is done simply by checking the dictionary. With no network delays, failure detection time and bandwidth usage will remains constant with the TCP protocol being used

Metrics







We ran tests to check the time taken for each command to complete and plotted a graph based on that. Since the time taken for each command remains approximately the same with increasing number of nodes in the network, we can say that the algorithm is scalable.