CS 451 Project
Implementation of Cassandra

Sagar Jha - 110100024
Rohan Gyani - 110040001
Abhishek Gupta - 110040067

May 5, 2015

1 Problem Description

We implement a key value store with two operations : insert and get.

Insert : Takes as input table name, key and value. Value is in the form supercolumn : column, where we identify supercolumns as a group of columns. Get : Takes as input table name, key and supercolumn name. Returns the set of values (the columns)

We also implement a messaging application that makes use of the cassandra system to store messages. It offers two functionalities other than send.

Term Search : Search for a keyword. Returns all messages that contains the keyword in time-sorted order.

Interaction Search : Search for interactions with a user. Returns all messages that were sent to this user in time-sorted order.

We use the messaging application for performance and scalability testing.

Performance Testing : Populate the table with messages and then compute the average time taken for insert and get.

Scalability Testing : We control the number of users and the amount of interactions between them and check the variability in average times for insert and get.

2 System Design

Storage at servers : For a given table, key and supercolumn, we have a unique file on disc, path of which is ¡table¿/¡key¿/¡supercolumn¿. Each such file stores all the values. We write sequentially to the file and hence the data is sorted by time of write.

We store the records in memory till a certain number of records before flushing to disc.

On disc the format for storage is $< column - name >< size >< column - name >< size > ...$

In memory, the data structure is an arrayList of keys and for every key, we have a map of supercolumns with the list of columns.

Key-space partitioning : Chord's algorithm is used for partitioning and replication. If a key falls into the domain of a certain node, that node is called the coordinator of the key. If k is the replication factor, the coordinator is responsible for storing it in its $k-1$ successors.

On an insert, we first direct the operation to the coordinator of the key. The coordinator sends insert commands to all the $k-1$ successors also. We perform synchronous writes and acknowledgement is send only after all the k replicas are stored.

On a get, we check if the node is one of the holders of the key. If not, then we direct the operation to the coordinator of the node. So, a get operation uses only one of the nodes that has the data.

3 Environment for execution

We require JVM. We use RMI heavily for communication. We will test the system on Linux Ubuntu.

4 Performance and Scalability (Optional)

This is just as desribed in the first section. We are not sure, if we will be able to perfomance these tests.