Design Document

Communication between System and Evaluation Program:

The communication between the distributed key/value systems which are to be tested (MonoDB, Cassandra, Redis, Riak & MyDHT) and the evaluation program happens through various APIs, specifically, JAR packages in Java. I have created a separate Java class file for each system. That Java class file makes use of JAR packages to connect to the cluster of instances on which the distributed key/value systems are running.

We'll go through the design of the system by referring each distributed key/value systems -

1. **MongoDB**
   a. MongoDB for running properly requires three configuration servers and one query router. Three configuration servers are required so that the system can work even if one configuration server goes dead. So, for MongoDB, I created three configuration server (instances), one query router (instance) and 16 cluster instances.
   b. ***connect(host, port)*** – This method connects to the query router using the IP address and Port number of the MongoDB instance running on query router. We send requests to the query router and it forwards it to the appropriate cluster node after hashing. This functionality is also called sharding.
   c. ***insert(key, value)*** – This method inserts a <Key, Value> pair in the distributed key/value store. Parameters required are Key and Value in string format.
   d. ***lookup(key)*** – This method searches for the appropriate key in the distributed key/value store and returns the value of the <Key, Value> pair if the specified key is present else returns null.
   e. ***remove(key)*** – This method removes the <Key, Value> pair for the specified key from the store. It doesn't give any error in case the key is not present.
   f. Note that hashing of key and storing the <Key, Value> pair on the appropriate node after hashing id done by the query router.

2. **Cassandra**
   a. Casandra was evaluated in a cluster of 16 instances. In order for these 16 instances to act as a cluster, each node in Cassandra has a parameter called seed node. Basically, all 16 instances have the same seed node in their configuration file. This helps them to discover their neighboring nodes using gossip protocol so that they can form a cluster.
   b. ***connect(host, port)*** – This method connects to one of the nodes in the cluster using the IP address and Port number of the Cassandra instance running on that node.
   c. ***insert(key, value)*** – This method inserts a <Key, Value> pair in the distributed key/value store. Parameters required are Key and Value in string format.
   d. ***lookup(key)*** – This method searches for the appropriate key in the distributed key/value store and returns the value of the <Key, Value> pair if the specified key is present else returns null.
   e. ***remove(key)*** – This method removes the <Key, Value> pair for the specified key from the store. It doesn't give any error in case the key is not present.
   f. ***disconnect()*** – This method disconnects from the node to which it is connected.
   g. Note that distribution of <Key, Value> pairs among the nodes in the cluster is done by the Cassandra system itself.

3. **Redis**
    a. Each Redis node comprises of two instances (two systems) one master and other acts as slave. This is necessary for Redis to work because even if the master fails, the slave is able to provide the data. So, it basically provides excellent replication. I created 32 instances to create a cluster of 16 nodes since each node requires two instances.
    b. ***connect(<Set of host, port>)*** – This method connects to the Redis cluster by taking a set of IP addresses and Port numbers of all the nodes in the cluster.
    c. ***insert(key, value)*** – This method inserts a <Key, Value> pair in the distributed key/value store. Parameters required are Key and Value in string format.
    d. ***lookup(key)*** – This method searches for the appropriate key in the distributed key/value store and returns the value of the <Key, Value> pair if the specified key is present else returns null.
    e. ***remove(key)*** – This method removes the <Key, Value> pair for the specified key from the store. It doesn't give any error in case the key is not present.
    f. ***disconnect()*** – This method disconnects from the Redis cluster.
    g. Note that distribution of <Key, Value> pairs among the nodes in the cluster is done by the Redis system itself. Redis doesn't perform consistent hashing. Instead it has a 16384 slots which is divided into the number of nodes in the cluster. So, each node has a slot range and after hashing the key, the <Key, Value> pair is inserted into the appropriate slot (or node since each nodes has a slot range).

4. **MyDHT**
    a. MyDHT refers to the Distributed Hash Table system designed and developed in Assignment 2. I created a cluster of 16 nodes (instances). Each instance runs a server as well as client.
    b. ***connect(fileName)*** – Path of the file which contains the list of IP addresses of all the nodes in the cluster.
    c. ***insert(key, value)*** – This method inserts a <Key, Value> pair in the distributed key/value store. Parameters required are Key and Value in string format.
    d. ***lookup(key)*** – This method searches for the appropriate key in the distributed key/value store and returns the value of the <Key, Value> pair if the specified key is present else returns null.
    e. ***remove(key)*** – This method removes the <Key, Value> pair for the specified key from the store. It doesn't give any error in case the key is not present.
    f. ***disconnect()*** – This method disconnects from the Riak cluster.
    g. MyDHT performs the hashing on the key and forwards the request to store the <Key, Value> pair to the appropriate node after hashing. Hashing determines that the <Key, Value> pair will be stored on which node.

5. **Riak**
    a. Riak was evaluated in a cluster of 16 instances. Initially, these 16 instances (nodes) are individual nodes without any clustering. We use a join command to join these instances thereby creating a cluster. After clustering, Riak divides all the data equally among the nodes and performs sharding as well.
    b. ***connect(fileName, port)*** – This method connects to the Riak cluster by taking input as port number and path to the text file containing IP addresses of all the nodes in the cluster.
    c. ***insert(key, value)*** – This method inserts a <Key, Value> pair in the distributed key/value store. Parameters required are Key and Value in string format.
    d. ***lookup(key)*** – This method searches for the appropriate key in the distributed key/value store and returns the value of the <Key, Value> pair if the specified key is present else returns null.

e. ***remove(key)*** – This method removes the <Key, Value> pair for the specified key from the store. It doesn't give any error in case the key is not present.
f. ***disconnect()*** – This method disconnects from the node to which it is connected.
g. Note that distribution of <Key, Value> pairs among the nodes in the cluster is done by the Riak system itself

The professor said that the design document is not required. I had completed till here, so I didn't write further part due to time constraints.