Mini Project
5th Semester
Chord Distributed Hash Table

Design and develop a application for implementing Chord DHT :-

1. A **distributed hash table** (**DHT**) is a class of a decentralised distributed system that provides a lookup service similar to a hash table: (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

2. In computing, **Chord** is a protocol and algorithm for a peer-to-peer distributed hash table. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key. Chord is one of the four original distributed hash table protocols, along with CAN, Tapestry, and Pastry.

3. Read the paper to implement chord.
   https://drive.google.com/file/d/0ByssYdG6YTB4NHFnN29GdTZTcFE/view?usp=sharing

4. **Protocol :-**
   The Chord protocol and algorithms are described in the paper [2]. You should read the paper to learn about the design of Chord, prior to starting your own implementation. The implementation in the paper, shown in Figure 5 and Figure 6, is presented as pseudocode. The pseudocode involves both local and remote function call invocations, determined by the node at which a function call is invoked.
   Note that this pseudocode is extended by Section IV.E.3 “Failure and Replication”, with changes to the ‘stabilise’, ‘closest preceding node’, and ‘find_successor’ function calls. Additionally, there is a new function call ‘get successor list’ hinted at in the description, which returns a node’s list of successors and is used in the extended function calls. We are providing the GPB protocol message types for you, which may be found in the ‘assignment5’ directory in the ‘materials’ repository. For this project, you may safely ignore the following portions of the paper:
   1. Section IV.E.4 “Voluntary Node Departures” - All node departures will be treated as node failures.
   2. Section II “Related Work”, Section V “Evaluation”, Section VI “Future Work”, Section VII “Conclusion”.

   3. **Chord Client:-**
      The Chord client will be a command line utility which takes the following arguments:
      1. -a = The IP address that the Chord client will bind to, as well as advertise to other nodes. Represented as an ASCII string (e.g., 128.8.126.63). Must be specified.
      2. -p = The port that the Chord client will bind to and listen on. Represented as a base-10 integer. Must be specified.
      3. –ja = The IP address of the machine running a Chord node. The Chord client will join this node’s ring. Represented as a ASCII string (e.g., 128.8.126.63). Must be specified if (–jp) is specified.
      4. –jp = The port that an existing Chord node is bound to and listening on. The Chord client will join this node’s ring. Represented as a base-10 integer. Must be specified if (–ja) is specified.

5. –ts = The time in milliseconds between invocations of ‘stabilise’. Represented as a base-10 integer. Must be specified, with a value in the range of [1,60000].
6. –tff = The time in milliseconds between invocations of ‘fix_fingers’. Represented as a base-10 integer. Must be specified, with a value in the range of [1,60000].
7. –tcp = The time in milliseconds between invocations of ‘check predecessor’. Represented as a base-10 integer. Must be specified, with a value in the range of [1,60000].
8. -r = The number of successors maintained by the Chord client. Represented as a base-10 integer. Must be specified, with a value in the range of [1,32].

An example usage to start a new Chord ring is:
chord -a 128.8.126.63 -p 4170 –ts 30000 –tff 10000 –tcp 30000 -r 4

An example usage to join an existing Chord ring is:
chord -a 128.8.126.63 -p 4171 –ja 128.8.126.63 –jp 4170 –ts 30000 –tff 10000 –tcp 30000 -r 4

The Chord client will open a TCP socket and listen for incoming connections on port (-p). If neither (–ja) nor (–jp) are specified, then the Chord client starts a new ring by invoking ‘create’. The Chord client will initialise the successor list and finger table appropriately (i.e., all will point to the client itself).
Otherwise, the Chord client joins an existing ring by connecting to the Chord client specified by (–ja) and (–jp) and invoking ‘join’. The initial steps the Chord client takes when joining the network are described in detail in Section IV.E.1 “Node Joins and Stabilisation” of the Chord paper.
Periodically, the Chord client will invoke various stabilisation routines in order to handle nodes joining and leaving the network. The Chord client will invoke ‘stabilise’, ’fix_fingers’, and ‘check predecessor’ every (–ts), (–tff), and (–tcp) milliseconds respectively.

Commands
The Chord client will handle commands by reading from stdin and writing to stdout. There are two command that the Chord client must support: ‘Lookup’ and ‘PrintState’. ‘Lookup’ takes as input an ASCII string (e.g., ”Hello”). The Chord client takes this string, hashes it to a key in the identifier space, and performs a search for the host that is the successor to the key (i.e., the owner of the key). The Chord client then outputs the IP address and port information of the host. ‘PrintState’ requires no input. The Chord client outputs its local state information at the current time, which consists of:
1. The Chord client’s own node information.
2. The node information for all nodes in the successor list.
3. The node information for all nodes in the finger table
   where “node information” corresponds to the identifier, IP address, and port for a given node.
   An example sequence of command inputs and outputs at a Chord client is shown below. There are four participants in the ring (including the Chord client itself) and -r is set to 2. You may assume the same formatting for the input, and must match the same formatting for your output. “>” and “<” represent stdin and stdout respectively. The input commands will be terminated by newlines (‘\n’).

   >Lookup Hello
   <Hello f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0
   <31836aeaab22dc49555a97edb4c753881432e01d 128.8.126.63 2001
   >Lookup World
   <World 70c07ec18ef89c5309bbb0937f3a6342411e1fdd
   <7d157d7c000ae27db146575c08ce30df893d3a64 128.8.126.63 2002
   >Print State
   <Self 31836aeaab22dc49555a97edb4c753881432e01d 128.8.126.63 2001

<Successor[1] 6fa8c57336628a7d733f684dc9404fbd09020543 128.8.126.63 2002
<Successor[2] 7d157d7c000ae27db146575c08ce30df893d3a64 128.8.126.63 2003
<Finger[1] 6fa8c57336628a7d733f684dc9404fbd09020543 128.8.126.63 2002
<Finger[2] 6fa8c57336628a7d733f684dc9404fbd09020543 128.8.126.63 2002
...
<Finger[159] 7d157d7c000ae27db146575c08ce30df893d3a64 128.8.126.63 2003
<Finger[160] f4d60480373006cb24147cd17765000f14aadca3 128.8.126.63 2004

5. General Instructions

1. Use any object oriented language for implementation.
2. The system should be able to compile on linux and windows.
3. For creating design use Umbrello UML or Argouml.