Distributed Systems Project 3
Pastry Protocol

1. Group Details:

Members

1: Manjary Modi ,UFID: 38408368, manjary.modi@ufl.edu
2: Rameshwari Obla Ravikumar,UFID: 16161302, rameshwari.oblar@ufl.edu

Steps to run the application:

1. In shell, navigate to the directory where mix.exs exists
2. Run command
>mix escript.build
[Ignore any warnings that you notice]
>./project3 <numNodes> <numRequests>
<numNodes> can be any number
<numRequests> can be any number
Example: ./project3 512 3
After step2, the program will start running based on the number of nodes(peers) and the number of requests provided and it terminates when all peers send out message(or reach) to the key node(destination).
3. The average number of hops traversed to reach the destination is printed on the screen before the program exits.

Application Details:

Our application consists of two ex files – project2.ex and pastry.ex
• project2.ex contains the main module which is used for getting the required inputs from the user, calls pastryInit (sets up the peer to peer network) and creates the routing table and theleaf sets
• pastry.ex contains the actual implementation for creating the nodes, binding a node with another, and message passing mechanism(routing algorithm).

[Explanation for each module and method is written further in the document]

2. What is working?

1. Pastry node creation

When the user provides number of nodes to be created in the peer to peer pastry network, our application creates required number of pastry nodes and encodes the processID into md5 hash so that the resulting hash(nodeID) containing only hex digits is 128-bit in length and the nodeIDs are uniformly distributed in the network and as a result, they are geographically of random distances from each other without any bias. The application then initializes the states of each pastry node – leaf set (set of pastry nodes that have nodeIds numerically closer to the current pastry node), routing table set (a list of list containing nodeIds with certain number of matching prefixes)

For example: The state of a pastry node contains the following information:

%{count: 0,
 leaf_set: %{larger: [], smaller: [""]},
 neigh_list: [], nodeId: "B8C9D8AE4B0EF96EAF9CC71F1760817E", pid: #PID<0.77.0>,
 routing_set: [], t_neigh: []}

2. Node joining

When a new node wants to join the network, it finds a node in the pastry network which is numerically closest to it and sends a message. The old node introduces the new node into the system by sharing its leaf set to the new node. The routing table rows of the new node are populated by using information from other nodes in the network.

3. Genserver for creating and maintaining the peer to peer system.

The main method calls the Genserver init method in pastry.ex for creating the pastry nodes and setting the states – leaf set, routing table and count of numRequests. Genserver call method is used for setting the state and Genserver cast method is used to asynchronously send messages from each of the peer. The cast method allows us to send one request/second from each peer. The cast method is made recursive so as to achieve parallel processing of requests.

4. Setting Routing Table for each node

The routing table in our pastry nodes is stored as a list of list, where each inner list represents the rows of the routing table. The dimension of the routing table are (logN base 16) x 16. Number of columns is taken as 16 (2^b, where b=4). The rows are implemented in decreasing order of the highest matching prefix(contrary to what is described in paper), that is, the first row contains the nodeIds that share the maximum number of matching prefix with the current node and the last row contains nodeIds that share no matching prefixes of the current node. The routing table is used to lookup the nodeIds that form the intermediate hopping destination when the initial(current) node searches for the ‘key’ destination.

5. Message Passing

When a node wishes to pass a message with a key(destination), it simply looks up its leaf set. If the key lies between the smallest of the smaller leaf set of sender and the largest of the larger leaf set of sender, then the destination ID is taken from the leaf set. If not, the sender’s routing table is used. It looks up the table to find a node that is closer to the key by a matching prefix larger than it’s own and makes the hop. The hopping continues till the key is reached. At every hop, the sender gets closer to the destination by one level. Hence, the message passing happens in a logarithmic time.

6. Average Hops

Hops refer to the number of lookups a pastry node performs to reach the destination node. Total number of hops is calculated by aggregating the number of hops each pastry node takes to perform ‘numReq’ times the requests, and then dividing by total number of message requests transmitted in the pastry network.

Average Hops = [Sum(No. of hops performed by each pastry node message request) x numRequests] / [numNodes x numRequests]

We noticed that the number of hops increases logarithmically with the total number of requests that gets transmitted over the network. The Data is as below

No.of Nodes	No. of Requests	Total Message	Average Hops
16	4	64	1.359375
64	4	256	1.68359375
256	4	1024	2.1796875
512	4	2048	2.408691406
1000	4	4000	2.5115

Explanation for each of the function used in project3.ex

1. main

→ Creates the pastry node by calling pastryInit(modelled as Genserver init in our application)
→ Sorts the pastry nodes so as to find numerically closest nodeIds
→ Sets the leaf list for each of the pastry nodes by calling Project3.leaf_setting()
→ Sets the routing table for each of the pastry node by calling Project3.routingSetting()
→ Propagates the message passing from each of the pastry node, as required

2. getProcessId

→ Returns the processId associated with the pastry node.
→ This is required for Genserver call/cast methods

3. leaf_setting

→ For each pastry node, it finds 4 numerically closest nodes that is larger and smaller than the current node and stores them in smaller and larger sublists of the leaf set.

4. extractNodeId

→ To extract the nodeIds (hashed output of processIds) from the initial node list.

5. routingSetting

→ Sets the routing table by calling Project3.set_routingTable
→ Calls the Genserver's call method set_routingSet to initialize the routing table state of a current pastry node .

6. set_routingTable

→ Sets each row of the routing table by calling set_routingTableROW

7. set_routingTableROW

→ Populates each row of the routing table by using matching prefixes.
→ For example, the last row contains 0 matching prefix and the first row contains maximum number of matching prefix for a given pastry node..

Explanation for each of the function used in pastry.ex

1. start_link

→ Calls the Genserver init method

2. init

→ Calculates 128-bit hash value for a process ID. We are using md5 algorithm to create the hash values
→ Initialize the state of a pastry node

3. setcount
→ Sets the state of count for each pastry node.
→ Implemented through a genserver handle call

4. get_state
→ Gets the state of a pastry node
→ Implemented through a genserver handle call
5. set_leaf
→ Sets the leaf set to a pastry node's state

6. set_routingSet
→ Sets the routing set to a pastry node's state

7. pickNode
→ When a node wishes to pass a message with a key(destination), it simply looks up its leaf set. If the key lies between the smallest of the smaller leaf set of sender and the largest of the larger leaf set of sender, then the destination ID is taken from the leaf set.
→ It looks up the table to find a node that is closer to the key by a matching prefix larger than it's own and makes the hop.
→ The hopping continues till the key is reached via the recursive call of pickNode. At every hop, the sender gets closer to the destination by one level.

8. genPickNode
→ Used for parallel message passing of each pastry node.
→ Implemented through a Genserver handle cast.

3. What is the largest network you managed to deal with?

1000 nodes with 4 requests (Totally, 4000 messages routed through the pastry network).
Average hops came out to be be 2.51 and the total number of hops was 10046.

We are sure our implementation works fine for larger messages than 4000. Since we are sending a message request per second, the sleep time used in the program increases the time required to run the application. Because of this time constraint, we were not able to test it for larger networks.