Distributed Systems

Pastry Protocol

READ ME file for Distributed Operating Systems - Project 3

Date: October 23, 2017

Group members:

1. **Mohit Mewara**, UFID: 8413-2114, mohitmewara@ufl.edu
2. **Nakshatra Sharma**, UFID: 4935-2902, nakshatra2312@ufl.edu

Project Execution Instructions:

For Pastry Protocol:

mix escript.build

./project3 numNodes numRequests

- Where numnodes is the number of peers to be created in the peer to peer system
- Where numRequests is the number of requests each peer must make

Implementation Details:

In this project we have implemented self-organizing and self-adaptive Pastry API, with dynamic network join and route methods, that can be used to provide services for implementing Distributed Hash tables, peer discovery in a peer-to-peer network, group communication, naming and global storage.

Project was implemented as per the specifications in the research paper Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.

In the project, we have created a distributed peer to peer network where each peer is associated with a unique GUID key. The GUID keys are generated by applying md5 hash function on the node number followed by encoding with base 16. Since we are encoding the keys with base 16, each character of the GUID can be represented with 0-15 bits.

As the program starts, it creates a self-adaptive prefix map containing all the possible prefixes as the keys and the nodes associated to that prefix as values. When a new node gets added to the network the map gets updated. Each node maintains a neighbor map which is filled by flipping the bits of the node's GUID one at a time to generate all possible prefixes. The nodes corresponding to that prefixes are stored as values by looking up the prefix map.

Once a node receives a message with destination key other than itself, it looks up in its neighbor map to pick the neighbor with longest prefix matching with the destination node and delivers the message to it.

The process is repeated until the message is delivered to the destination node or no prefix is matched in the neighbor map.

Actor Modelling:

We have used actor modeling to create the nodes and parallelly send the message requests to its random neighbors.

The program exits when all the peers performed that many requests and the average number of hops that must traversed to deliver a message is printed on the screen.

Output:

The output of the command "escript project3 2 2" is discussed below:

- **Start**: GUID of the node that is initiating the message send request.
- **Final**: GUID of the destination node.
- **Found**: GUID of the node at which the message is not delivered further. It may be the Found node or any other node if no neighbor is found with prefix match.
- **Hop**: Number of nodes in between start and Found.

The final and found nodes are same in the below table which means that our **message is successfully reaching the destination node**.
The average Hop count is **0.75**.

Start	Final	Found	Hop
CFCD208495D565EF66E7DFF9F98764DA	CFCD208495D565EF66E7DFF9F98764DA	CFCD208495D565EF66E7DFF9F98764DA	0
C81E728D9D4C2F636F067F89CC14862C	CFCD208495D565EF66E7DFF9F98764DA	CFCD208495D565EF66E7DFF9F98764DA	1
CFCD208495D565EF66E7DFF9F98764DA	C4CA4238A0B923820DCC509A6F75849B	C4CA4238A0B923820DCC509A6F75849B	1
C4CA4238A0B923820DCC509A6F75849B	CFCD208495D565EF66E7DFF9F98764DA	CFCD208495D565EF66E7DFF9F98764DA	1

The table below shows the command with different number of nodes and requests.

Number of Nodes	Number of Requests	Average Hop Count	$\log_{16} n$
10	10	1.12	~1
10	1	1.14	~1
100	10	1.82	<2
100	1	1.75	<2
1,000	10	2.67	<3
1,000	1	2.56	<3
10,000	10	3.45	<4

100,000	10	4.32	<5
100,000	1	4.12	<5

Result:

The project was tested for 100,000 nodes and the average hops as 4.1

Since we are encoding the GUIDs with base 16, the average hops for routing of message from sender to receiver has the worst case complexity of O(log16 n), where n stands for the number of nodes in the network. This is because with each character prefix matched the number of hop nodes get reduced with a factor of 16.
The complexity appears to be logarithmic irrespective of the number of requests, proving that all the requests are being sent parallelly.