# CS425

Spring 2015

MP2

# Chord Peer-to-Peer System

Hao Jin & Yisong Yue

haojin2 & yyue7

## 1 Introduction

Chord in a kind of implementation of DHT (Distributed Hash Table), for MP2, we are required to implement such a system using the algorithm in the paper provided [1]. The keys and nodes are all represented by some identifier that lies on the hash ring, which has 256 slots for both keys and nodes. All keys are assumed to be present, each slot on the hash ring can hold up to 1 node.

## 2 Data Sturcture for Each Node

Each node keeps record of the keys that are stored inside itself using an ordered map from the standard library of C++. Each node is also responsible for recording its own predecessor and identifier. There are two arrays in each node, one is stores the nodes that fingers points to , the other simply holds the starting point for each finger table entry. The values for the starting point of each finger table entry are calculated upon initialization of each node.

## 3 Communication between Nodes

All nodes, including the coordinator thread, communicate with each other using sockets. All connections are created and used when needed, and closed after usage. Ports used are determined by the nodes' identifiers.

## 4 Structure of Messages

There is a class defined for messages in the network, each messages carries information such as message type and other information needed.

## 5 Coordinator

The coordinator is not one of the nodes on the hash ring, it initializes the system upon launch and adds node 0 onto the hash ring to be the very first node in the system. Then it waits for the user to enter prompts and perform corresponding tasks.

[1] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications[J]. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149-160.

## 6 Nodes

Each node waits for any connect() called onto its own port from the coordinator or other nodes, after establishment of the connection, it reads in the message that is sent to it, then it decodes and handles the message. After processing the message, it sends back corresponding reply.

## 7 Performance Test Data and Analysis

$N = 10$ fro all values of $P$, $F$ is generated randomly with $F \geq 64$ guaranteed for each test.

| $P$ | $\overline{Join}$ | $\overline{Find}$ |
|---|---|---|
| 4 | 68.65 | 6.716 |
| 8 | 87.2 | 7.866 |
| 10 | 93.3 | 8.337 |
| 20 | 115.1 | 10.181 |
| 30 | 126.3 | 11.440 |

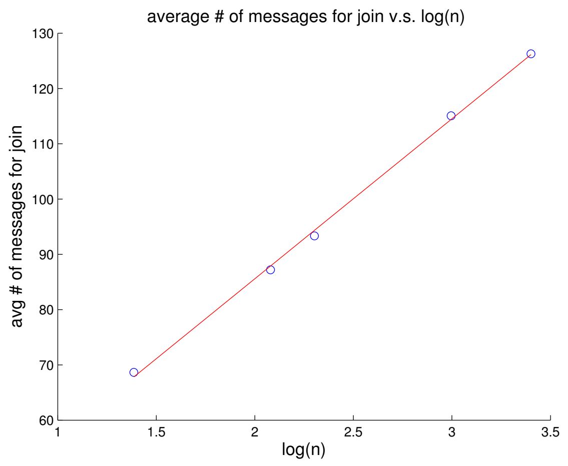

Figure 1: $\overline{\# \, of \, messages \, for \, a \, single \, join} \, v.s. \, log(n)$

Fitting parameters:

| $R^2$ | Adjusted $R^2$ | RMSE |
|---|---|---|
| 0.9987 | 0.9982 | 0.9571 |

The results of the curve fitting suggests that for a single join operation, average messages exchanged is proportional to $log(n)$ where $n$ is the number of nodes in the system. So it takes $O(log(n))$ time for a single join on average.

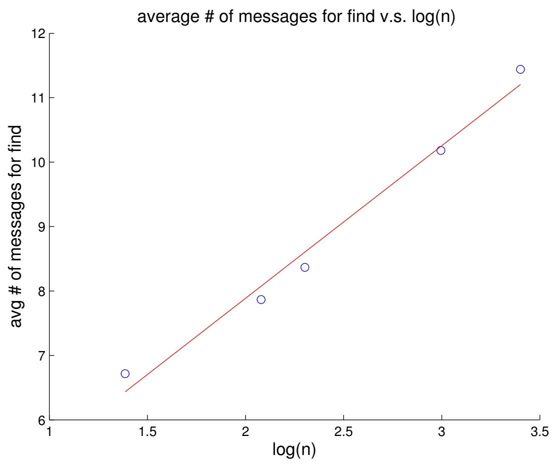

Figure 2: $\overline{\#\ of\ messages\ for\ a\ single\ find}\ v.s.\ log(n)$

Fitting parameters:

| $R^2$ | Adjusted $R^2$ | RMSE |
|---|---|---|
| 0.9832 | 0.9777 | 0.2817 |

The results of the curve fitting suggests that for a single find operation, average messages exchanged is proportional to $log(n)$ where $n$ is the number of nodes in the system. So it takes $O(log(n))$ time for a single find on average.

## 8 Conclusion

The system by our group perfectly produce the ideal consequence for find operation, and the performance for join operation is slightly better than the $O(log^2(n))$ theory in the paper. We think this is due to the fact that the size of the system grows from 1 to n when the nodes join.