Project 3 Proposal and Final Report

Name: Tianqi Wen, Mengyu Yang
Andrew ID: tianqiw, mengyuy

Proposal

The idea of this project came from the paper *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications* [1], which is presented by one of the group member during the paper presentations session. The paper introduces *Chord*, a distributed lookup protocol that solves the problems of efficient location of the data node. Chord adapts efficiently as nodes join and leave the system and can answer queries even if the system is continuously changing.

The paper presents the algorithms for (1) stabilization, (2) scalable key lookup using the finger table and (3) failure and replication. In project 3, we want to implement these three algorithms and simulate the behavior of Chord. The algorithms will be implemented as Java program.

Structure

```
MainTest: read and deal with user command
ChordRing ------> ChordNode ------> HashKey
                            |---> FingerTable ----> Finger
Hash: calculating hash using SHA-1
```

What we have done

(1) Stabilization:
The join and stabilization are implemented according to the pseudo code of paper *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications*. The method is shown as below:

// join the chord ring with new node n’
ChordRing.**join**(ChordNode n’)

// called periodically to check if a new node has been add to n’s successor
// also deal with failed successor
n.**stabilize**()

// node n’ notify n that it might be n’s predecessor
// used when new node is added or old node failed
n.**notify**(n’)

```
    // refresh finger table
    fix_fingers()
```

(2) Scalable key lookup using the finger table:

Finger table is implemented using an ArrayList with type Finger. The ith finger maintains the start hash key with number (hash(original_node) + 2^{i}-1) and its successor.

(3) failure and replication

Instead of only maintaining the direct successor and predecessor, the ChordNode also maintains its 2nd successor (the number can be bigger in practical implementation). When encountered failure, although direct successor information might be lost (when the failure take place the node leaves chord ring does not notify its predecessor and successor), it can use the 2nd successor to re-connect the ring.

How to test

The code has been tested on Windows 8 and shark machines (Linux). All the command keywords are case-insensitive.

To run the code:

(1) Unzip the chord.zip and type **javac *** to compile all the java files.

(2) **java MainTest <Max node number> <Initial node number> <Lookup method>**

Lookup method can be one of (1) simple or (2)scalable.

For example: java MainTest 128 16 scalable

The program will print out 16 nodes with IP address, port number and corresponding hash key.

(3) Add one node by typing **join <Identifier>.** The system will calculate the hash value of the identifier and insert this new node into the ring according to the lookup method you defined.

(4) Simulate the node leaving behavior by typing **leave <Identifier>.**

This method simulates accidental node failure. When leave is called, the nodes in the ring will not be notified that its successor/predecessor has left the chord ring. But using stabilization and replication, the chord ring is able to be reformed after the failure.

(5) To lookup node B from node A, type: **lookup <A's Identifier> <B's Identifier>.** The program will print out the tracing path from A to B. This simulates when a node A in the p2p system wants to lookup node B.

(6) **dump**: show all the information in the chord ring

(7) **fingertable <Identifier>** or **ft <Identifier>**: show the finger table content

(8) To exit the program: type **exit**.

Example Input/Output

1. **java MainTest 64 16 simple**

```
8 nodes are created.
Node ID: http://192.168.56.1:9007, Hash Key: 17c0
Node ID: http://192.168.56.1:9001, Hash Key: 3812
Node ID: http://192.168.56.1:9005, Hash Key: 4a06
Node ID: http://192.168.56.1:9006, Hash Key: 766b
Node ID: http://192.168.56.1:9002, Hash Key: a98c
Node ID: http://192.168.56.1:9003, Hash Key: b794
Node ID: http://192.168.56.1:9004, Hash Key: cf11
Node ID: http://192.168.56.1:9000, Hash Key: ed5d
17c0(4a06)==>3812(766b)==>4a06(a98c)==>766b(b794)==>a98c(cf11)==>b794(ed5d)==>cf
11(17c0)==>ed5d(3812)==>(to the head)
17c0==>ed5d==>cf11==>b794==>a98c==>766b==>4a06==>3812==>(to the head)
```

The first line shows the ring iteration in successor direction, in the “()” shows the second succssor.
The second line shows the iteration in predecessor direction

2. **join 123**

```
N69ed joins...
17c0(4a06)==>3812(69ed)==>4a06(766b)==>69ed(a98c)==>766b(b794)==>a98c(cf11)==>b
794(ed5d)==>cf11(17c0)==>ed5d(3812)==>(to the head)
```

hash(123) = 69ed
It is add between 4a06 and 766b.

3. **leave http://192.168.56.1:9007**

```
3812(69ed)==>4a06(766b)==>69ed(a98c)==>766b(b794)==>a98c(cf11)==>b794(ed5d)==>cf
11(3812)==>ed5d(4a06)==>(to the head)
```

hash(http://192.168.56.1:9007) = 17c0

4. **lookup http://192.168.56.1:9001 http://192.168.56.1:9000**

```
Identifier: http://192.168.56.1:9001, Hash Key: 3812
Lookup for Identifier: http://192.168.56.1:9000, Hash Key: ed5d
Tracing... http://192.168.56.1:9001
Tracing... http://192.168.56.1:9005
Tracing... 123
Tracing... http://192.168.56.1:9006
Tracing... http://192.168.56.1:9002
Tracing... http://192.168.56.1:9003
Tracing... http://192.168.56.1:9004
Node ID: http://192.168.56.1:9000, Hash Key: ed5d
```

The trace of looking up http://192.168.56.1:9000 is printed out, it takes 7 steps

5. **Change to "scalable" mode and run the same lookup**

```
Identifier: http://192.168.56.1:9001, Hash Key: 3812
Lookup for Identifier: http://192.168.56.1:9000, Hash Key: ed5d
Tracing... http://192.168.56.1:9001
Tracing... http://192.168.56.1:9005
Tracing... 123
Node ID: http://192.168.56.1:9000, Hash Key: ed5d
```

The trace is printed out. By applying scalable key location, the lookup only takes 3 steps.

Future Improvement

(1) Multi-threading

Currently the stabilize() runs in the main loop in MainTest.java. It will be better to run it in the background process.

(2) Interface for functional server/client

This project only simulates the lookup using chord, no real connection or file transfer has been established. Interface to servers/clients (e.g. the one we developed in Project 1) can be provided.

Reference

[1]https://blackboard.andrew.cmu.edu/bbcswebdav/pid-847669-dt-content-rid-5489230_1/courses/S15-14740/chord-ton.pdf