Programming Assignment 3 (MP3)
CS 425/ECE 428 Distributed Systems
Spring 2018
Due: 7:00 p.m. April 26, 2018

This MP must be performed by groups of two students each. Any exceptions must be approved by the instructor.

The purpose of this assignment is to implement a peer-to-peer lookup service based on the paper "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications" from the SIGCOMM 2001 conference.

In particular, the goal is to implement key insertion, key lookup (or key location) and node join procedures. You will find Sections 4 and 5 of the above paper relevant to this assignment.

Assumptions: You may make the following assumptions.

- Only one operation will occur at any time. For instance, while a key look-up is in progress, a new node will not join; similarly, while a new node is in the process of joining, a new key lookup will not be initiated.
- At most one node may fail at any time (i.e., another node failure will not occur until the previous node failure has been handled, and system state has stabilized).
- The node identifiers and keys both consist of 8 bits (i.e., in the range 0 to 255). We will not be hashing the keys or node identifiers (or, in other words, they are assumed to be already specified as hash values).
- The system consists of at most 32 nodes.
- Node 0 never fails.

Implementation

For achieving message-passing between nodes, you may use the code from previous MPs. In particular, you should be able to add delay in delivering messages, similar to previous MPs. Similar to previous MPs, the configuration file should specify the min and max delay to be added.

Initialize your system to consist of a single node, with identifier 0, with keys 0 through 255 initially stored at node 0.

Implement a client at node 0 that reads commands from the keyboard input and prints out a response. The client should connect to all the nodes. The client will propagate the user's commands to the appropriate node and then wait for a response, before executing the next input command.

Commands

The following commands may be input to the client. Client should contact the appropriate node, as specified in the command.

- join p – Add node p to the system. Create a new node with identifier p, which should then take steps to add itself to the network. Use node 0 in place of node n' in Section 4.4 of the above paper. When a new node joins, some of the keys will be transferred to the new node.

- find p k – If node p does not exist or has crashed, then the client should print "p does not exist". Otherwise, the client should send a message to node p asking it to initiate a procedure for finding key k. Upon finding location of key k, node p should print the identifier for the node that contains key k. Once node p has found the key and printed the above information, it should inform the client that the find procedure is complete. If key k does not exist, then node p will not be able to find it, and it should print "key k not found".

- crash p – The client should inform node p to perform a **clean** crash. Node p will stay crashed unless "join p" is executed later. When a node crashes, no additional message is sent by node p to any other node (for instance, node p should not inform others about its own crash). You will need to implement a *failures detector* for the other nodes to learn that p has crashed.

  When a node p crashes, any keys that are stored at node p will be lost.

- show p – print the identifier p, finger table at p, and the keys stored locally at p, if p exists in the system. If node p does not exist or has crashed, then print "p does not exist".

- show all – perform "show p" command for each node p that currently exists in the system.

Besides issuing the commands, the client should not participate in operations necessary to perform the above commands.

When each of the above operations is completed, the node p (specified in each command above) should inform the client. The client can then process the next command. This will ensure that two commands will not be processed simultaneously. (Note that a practical system needs to be able to handle overlapping operations, however, we will not handle such operations in this MP.)

Correct operation of "show p" and "show all" commands will be necessary to be able to evaluate your submission during the demo, and to test correctness of the other commands.

Format

The format for the output of the “show” commands, corresponding to any one node, should be as shown below, f1 is the first entry in the finger table and f_last is the last entry in the finger table, key1 is the smallest key stored at the specified node, and key_last is the largest. The keys should be printed as integers separated by a single space. “show all” should print such output for each node, in an increasing order of the node identifier.

```
node-identifier
FingerTable: f1, f2, f3, f_last
Keys: key1  key2  … key_last
```

Performance evaluation:

Repeat the following experiment N times without any node crashes, and report the average message overhead.

A single experiment is to be performed as follows:

- Phase 1: Add P nodes to the initial system, with their identifiers chosen randomly (taking care that a node that is already in the system is not added again).

  Count the total number of messages sent in Phase 1, and compute the average number of messages required for each node addition.

- Phase 2: Perform the find operation F times, with p and k chosen randomly, taking care that node identifier p is randomly chosen from the nodes that were added above. Any given key k may be potentially repeated during the F find operations.

  Count the total number of messages sent in Phase 2, and compute the average number of messages required for each find operation.

Run the experiment for N >= 5 (larger is better), for P = 4, 8, 10, 20, 30, and F = 128. Ensure that the different runs use a different seed for random number generation. Report the data in the following tabular format. In particular, provide the average number of messages in phase 1 and 2 for each value of P (averaged over all N runs for each value of P).

	Phase 1: Average number of messages	Phase 2: Average number of messages
P=4, N=?		
P=8, N=?		
P=10, N=?		
P=20, N=?		
P=30, N=?		

Report

Include the above table containing your performance data, and a brief discussion of any trends you observe in the data, and an explanation for the trends (no more than 10 lines, in 12 point single-spaced format).

Grading Rubric

If “show p” or “show all” command does not work, we will not be able to evaluate your MP properly, and hence the total grade for the MP will be limited to at most 30%, based on inspection of the code. When these commands work, the following rubric will apply:

- 5% data replication
- 20% join p command
- 20% find p, k command
- 20% crash p command
- 15% report
- 5% basic code cleanliness (good comments, reasonable organization and design).
- 15% questions during demo.