Programming Assignment 3: CHORD Distributed Hash Table

DUE Friday Dec. 1, 11.59.59 PM. NO EXTENSIONS.

1 Overview

This assignment is intended to familiarize you with distributed hash tables. You will implement the CHORD Distributed Hash Table, as described in lecture P2P systems and I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, SIGCOMM 2001. You will implement a dht-peer which interacts with other dht-peers to build a ring as described in lecture slide on P2P. You can use any SHA1 implementation as the consistent hashing function. Use IP addresses as identifiers for the nodes.

You will also implement a client dht-client that connects to the root of the DHT and submits requests to store objects, or requests to retrieve objects. Each request if for one object at a time. The objects are very small files, you can assume that they fit into a TCP packet.

1.1 Part 1 - DHT Peer

In the first part of the project, you will implement a DHT peer. A peer will join the DHT by contacting the root (this is the first peer to be started and its IP address will be know by any peer that wants to join).

Join. When joining, the new node will receive back what is its place in the ring, i.e. the successor and the predecessor in the ring. This will be determined as described in class lecture based on NodeID of node, where NodeID is obtained by applying a hash function on the IP address of the node. Note that other nodes in the ring will need to update their information to reflect the addition of the new node.

Leave/crash. A node can also crash. In this case the ring has to repare itself - nodes can periodically ping peers or just monitor the TCP connection and detect a link failure and fix the ring.

Lookup. The ring should implement two routing (lookups) protocols: (1) recursive and (2) iterative. Recursive is as discussed in class lecture, where the request for an object is propagated in the ring till it reaches the right node. Iterative is where the response for the next hop always goes back to the client and the client will then contact the next node continuing the request till the node where the object is stored is located.

Storage Each node needs to store on disk the received objects. You can have an index file where you store all objects.

Print the ring. To know that join and leave succeeded properly, you will need to check that the ring is well formed. After each join or leave, your peers should print information showing that the ring is well-formed. For example, each node can print it's successor.

1.2 Part 2 - DHT Client

A client connects to the root and either stores or retrieves objects.

Storing an object. To store an object, the client will obtain an ObjID by applying a hash function on the name of the object. Assume objects are small files (fit into a packet and names have a maximum size of 512 bits). The client contacts the root to obtain the NodeID of the node where it needs to store the object. Once the root figures out on what nodes the object needs to be

stored (propagate a request in the ring till finding the node with the right NodeID that needs to store the object), will return information about that node to the client, i.e. IP address and port, and then the client can contact directly the peer to store the object. IMPORTANT: the root does not know all the nodes in the ring, so information needs to propagate on the ring.

Retrieving an object. To retrieve an object, the client computes the ObjID, submits the request to root, root finds the node and the returns the information about that node to the client, then client contacts directly the node to obtain the object. Retrieve of an object can be done using the two lookup procedures specified above, iterative and recursive.

Menu. Once the client connects, a menu should be prompted with four options: store object, retrieve object iterative, retrieve object recursive and exit. Each option is selected by pressing a letter s for store, i for iterative retrieve, r for recursive retrieve and e for exit. Once an option is selected a message should be prompted asking for the object to be stored/retrieved (unless the option is exit). User introduces the object name. Message of success should be prompted when object is stored or retrieved, message of failure should be promoted for failure to store or retrieve. Once an operation is completed, the menu should be displayed again, your program should not exit unless the menu option for exit is chosen. For both success and failures your program should print on what node was the object stored (or retrieved).

2 Language

You can use C, C++ or Python for this project. Do not use libraries that are not installed by default on the CCIS Linux machines. Similarly, your code must compile and run on the command line. You may use IDEs (e.g. Eclipse) during development, but do not turn in your IDE project without a Makefile. Make sure you code has no dependencies on your IDE. You can use either multi-threading or events-based programming (i.e. select function).

NOTE 1: Your protocol design should be small overhead, i..e do not use strings for field headers that do not need to be strings and make unnecessary conversions between types.

NOTE 2: Use ports using the method from project 2 so you do not overlap with other colleagues. Leave 20 ports distance between you and your next colleague.

NOTE 2: Use scripts to launch your DHT-peers on several machines. You code should run with 5-7 peers, do not hardcode these numbers, i.e. your program should be general to ran with more, but test it with at least 5 peers.

3 Protocol

You are required to define your own protocol headers, please document them carefully. Things to consider for including in headers: source id and destination id, client id in order to be able to handle multiple requests at the same time, type of routing method used, object id. Do not use strings in your headers, unless needed, file names are ok, identifiers for nodes can be char buffers as outputted by the hash function.

Document in details the data structures used.

4 Your program

Your dht_peer program must execute on the command line using the following command.

```
$ ./dht_peer <-m type> <-p own_port <-h own_hostname> <-r root_port> <-R root_hostname>
```

The *-m type* parameter will specify if this peer is the root or not, 1 means root, 0, regular peer. Default should be 0, regular peer.

Examples:
To start the root

```
$ ./dht_peer -m 1 -p 3400 -h crown.ccs.neu.edu
```

To start a peer

```
$ ./dht_peer -p 3402 -h top.ccs.neu.edu -r 3400 -R crown.ccs.neu.edu
```

If your peer needs additional ports, you can chose them based on the port provided in the command line. For example, the root can listen on 3400 for peers and on 3400+10= 3410 for clients. You can chose the rule, make sure you do not overlap with other students. You are required to document all the ports used by your code.

Your dht_client program must execute on the command line using the following command.

```
$ ./dht_client <-p client_port> <-h client_hostname> <-r root_port> <-R root_hostname>
```

5 Other Considerations

Your programs must verify the validity of messages by strictly checking their format. If a received message is not as expected, such as an incorrect field or wrong message type, you must assert an error and terminate your program. You should be strict; if the returned message does not exactly conform to the specification above, you should assert an error. Remember that network-facing code should be written defensively.

6 Submitting Your Project

Follow post @39 from piazza, with name of the project "project3".

To turn-in your project, you should submit your (thoroughly documented) code along with three other files:

- A Makefile that compiles your code.
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, any challenges you faced, and an overview of how you tested your code.

Your README, Makefile, and source code, etc. should all be placed in a directory on a ccs account before submitting. You may submit as many times as you wish; only the last submission will be graded, and the time of the last submission will determine whether your assignment is late.

7 Grading

This project is worth 100 points, 70 points peer and 30 points client. All student code will be scanned by plagiarism detection software to ensure that students are not copying code from the Internet or each other.

In order to get full credit you will have to also explain how you tested your code, what scenarios did you consider, how do you know your code works correctly. You can include any documentation or code, state machine diagrams, wireshark screen terminals, testing code.

Testing will include:

- join a node
- leave for a node
- store an object
- retrieve an object with iterative lookup
- retrieve an object with recursive lookup

8 Extra Credit

Implement the finger table functionality described in class. This extra credit is worth 70 points.

9 Additional resources

You may find the following resources helpful

- Socket programming: `http://beej.us/guide/bgnet/`
- Unix programming links: `http://www.cse.buffalo.edu/~milun/unix.programming.html`
- C/C++ programming link: `http://www.cplusplus.com/`