Programming Assignment 1: Chord Distributed Hash Table

Due: March 8th, 2018 23:59:59pm

In this assignment, you will create a basic distributed hash table (DHT) with an architecture similar to the Chord system [1]. This assignment is worth **13.3%** of your total score in this course. You **must** work **by yourself** on this programming assignment and use **Python, C/C++, or Java**.

I have provided a file, `chord.thrift`, that defines several operations the DHT supports.

The assignment consists of four parts, which are described below.

1 Compile the interface definition file

I have provided an Apache Thrift interface definition file, `chord.thrift`. It can be downloaded from Blackboard. You can use Thrift to compile interface definition file to C/C++, Java, or Python.

I have compiled and installed Thrift under my home directory on CS Department computers. These computers are located in G7 and Q22, or you can also remotely access them by SSH'ing into `remote.cs.binghamton.edu`. Source code as well as compiled files are located under `/home/yaoliu/src_code/thrift-0.10.0`. Other relevant files can be found in `/home/yaoliu/src_code/local`. To use Thrift installed under my home directory, you need to set the environment variable PATH:

```
$> bash
$> export PATH=$PATH:/home/yaoliu/src_code/local/bin
```

Thrift can be used to create service stubs in a language of your choice from a `.thrift` interface definition file. As an example, the Thrift command to compile this assignment's IDL file to Java is as follows:

```
$> thrift -gen java chord.thrift
```

In `http://thrift.apache.org/tutorial`, you can find example clients and servers written in many of the languages that Thrift supports.

For your convenience, we have preprared example Thrift code, adapted from the Thrift tutorial, in C++, Java, and Python. We have also included relevant Makefile and bash scripts for setting the environment variables and compiling, and running the code on CS department computers. You can find them at `https://github.com/vipulchaskar/thrift-lab` . You should read the code in the language of your choice (one of Java, C++, and Python) before beginning work on the remainder of the assignment.

Information about the basic concepts and features of Thrift are available at `http://thrift.apache.org/docs/concepts` and `http://thrift.apache.org/docs/features`. Although I have provided the IDL file for you, you should also check `http://thrift.apache.org/docs/idl` for details about the Thrift interface definition language and data types and services supported by Thrift.

2 Extend the server-side method stubs generated by Thrift

You must implement methods corresponding to the following 6 server-side methods:

writeFile given a name, owner, and contents, the corresponding file should be written to the server. Meta-information, such as the owner, version, and content hash (use the **SHA-256 hash**) should also be stored at the server side.

If the filename does not exist on the server, a new file should be created with its version attribute set to 0. Otherwise, the file contents should be overwritten and the version number should be incremented.

If the server does not own the file's id, a SystemException should be thrown.

readFile if a file with a given name and owner exists on the server, both the contents and meta-information should be returned. Otherwise, a SystemException should be thrown and appropriate information indicating the cause of the exception should be included in the SystemException's message field.

If the server does not own the file's id, a SystemException should be thrown.

setFingertable sets the current node's fingertable to the fingertable provided in the argument of the function. I have provided initialization code (Part 3) that will call this function, but you need to correctly implement this function on the server side.

findSucc given an identifier in the DHT's key space, returns the DHT node that owns the id. This function should be implemented in two parts:

1. the function should call `findPred` to discover the DHT node that precedes the given id
2. the function should call `getNodeSucc` to find the successor of this predecessor node

findPred given an identifier in the DHT's key space, this function should returns the DHT node that immediately precedes the id. This preceeding node is the first node in the counter-clockwise direction in the Chord key space. A SystemException should be thrown if no fingertable exists for the current node.

getNodeSucc returns the closest DHT node that follows the current node in the Chord key space. A SystemException should be thrown if no fingertable exists for the current node.

To properly implement `findSucc`, `findPred`, and `getNodeSucc`, I highly recommend reading the Chord paper [1] to obtain a clear understanding of the Chord protocol.

According to the Chord protocol, each Chord node is responsible for a portion of the Chord key space **(see [1])**. For this assignment, a node's id is determined by the SHA-256 hash value of the string "<ip address>:<port number>". A file's id is determined by the SHA-256 hash value of the string "<owner>:<filename>". **Attempts to `writeFile` or `readFile` from a node that does not own the file's associated SHA-256 id should throw a *SystemException*.**

Note: When accessing `remote.cs.binghamton.edu`, you are actually redirected to one of the 8 `REMOTE` machines ({remote00, remote01, ..., remote07}.cs.binghamton.edu) using DNS redirection. So you need to make sure you use the public IP address of the remote machine that the server is running on. For example, the public IP of `remote01.cs.binghamton.edu` is `128.226.180.163`.

Example: If ten nodes are in the DHT running on 10 different ports on "`128.226.117.49:9090`", ..., "`128.226.117.49:9099`", and we want to write the a file "example.txt", owned by "guest", then the key associated with this file SHA256("guest:example.txt")="ad0c..." must be written to the node associated with the next SHA-256 value in the Chord id space. According to the Chord DHT specification (shown in Figure 1), this node is "`128.226.117.49:9093`', which has an SHA-256 hash of "c529...".

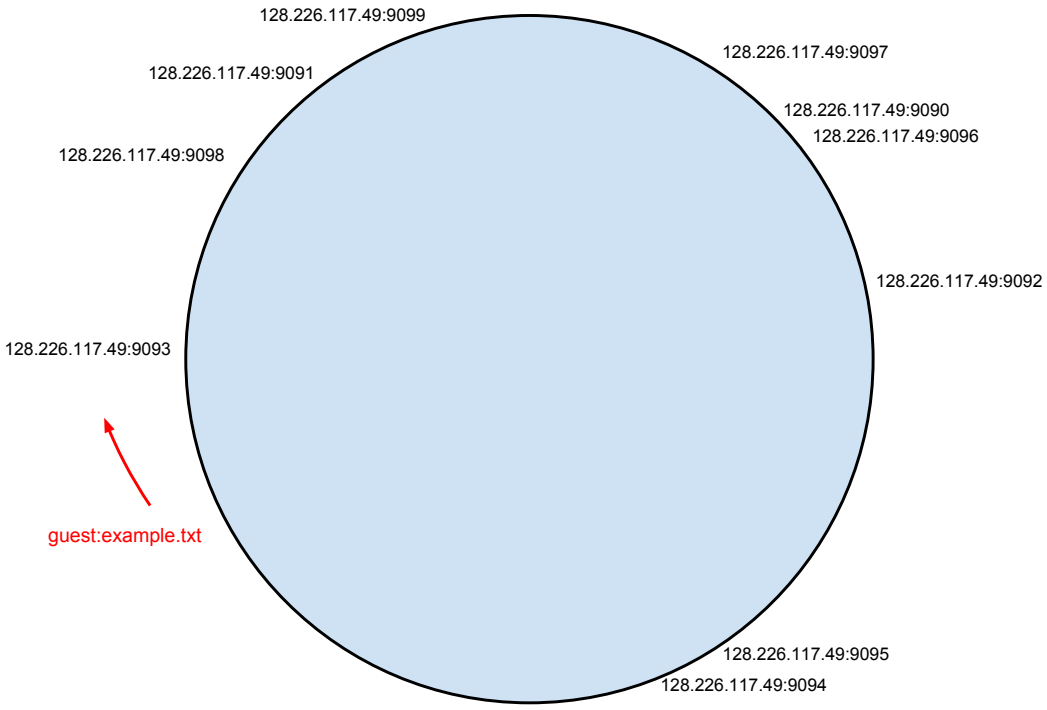

Figure 1: Example Chord Ring

Every time the server is started any filesystem or other persistent storage that it used should be initialized to be empty. If your server stores files in the file system, it should use only the current working directory from where it was run. There is no need to implement directory structure.

In addition, the server executable should take a single command-line argument specifying the port where the Thrift service will listen for remote clients. For example:

```
./server 9090
```

It should use Thrift's **TBinaryProtocol** for marshalling and unmarshalling data structures. Multiple servers will be running at the same time. Servers will also have to send RPC requests (e.g., `findPred` and `getNodeSucc`) to other servers.

3 Run the initializer program

For each DHT node, a fingertable needs to be initialized. In this assignment, I have provided a finger table initializer program that will compute the fingertable (see the Chord specification [1]) for each running DHT node and send each fingertable (using the `setFingertable` server method you implement) to the appropriate node. The initializer program can be downloaded from myCourses. The initializer program takes a filename as its only command line argument. To run the initializer:

```
$> chmod +x init
$> ./init nodex.txt
```

The file (`node.txt`) should contain a list of IP addresses and ports, in the format "<ip-address>:<port>", of all of the running DHT nodes.

For example, if four DHT nodes are running on `remote01.cs.binghamton.edu` port 9090, 9091, 9092, and 9093, then `nodes.txt` should contain:

```
128.226.180.163:9090
128.226.180.163:9091
128.226.180.163:9092
128.226.180.163:9093
```

The initializer program will print an error message if any of the specified DHT nodes is not available.

4 How to test

This is for testing your implementation only. You do not need to submit this part. It will not be graded.

To test your Chord-based file server implementation, you will need to write your own test program. This test program will act as a client that issues remote procedure calls to the Chord servers. It will then check if the returned values are correct.

Your test program need to verify the correctness of all six remote procedure calls supported by the server. For example, your test program might issue a `readFile` or `writeFile` call to a server that does not own this file's associated id. It should expect to catch a *SystemException*, thrown by the server.

To find the correct server to send `readFile` and `writeFile` requests to, your test program should first find the server that owns the key associated with the input user and filename: SHA-256("<user>:<filename>")[1] using the `findSucc` call. Then your test program can call `readFile` or `writeFile` on the server returned by the `findSucc` call.

5 How to submit

To submit the assignment, you should first create a directory whose name is "your BU email ID"`-p1`. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-p1`.

You should put the following files into this directory:

1. The `chord.thrift` file I provided.
2. The `init` initializer program I provided.
3. Your source code which includes at least one file implementing the server.
4. A `Makefile` to compile your source code into an executable, `server`. (It is okay if this executable is a bash script that calls the Java interpreter, as long as the command line argument follows the format described in Part 2.)
5. A `Readme` file describing the programming language(s)/tools (e.g., Apache Ant) that you used, how to compile and run your code on `remote.cs.binghamton.edu` computers, and anything else you want the TA to be aware of while grading your code.
6. A `STATEMENT` file, containing the following statement followed by the student's full name:

   "I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of **0** for the involved assignment and my grade will be reduced by one level (e.g., from

[1] SHA-256("<owner>:<filename>") indicates computing the hexadecimal string associated with the SHA-256 hash of a concatenation of the user and filename strings. For instance, if the user string is "guest" and the filename string is "example.txt", then the 64-character hexidecimal string of "guest:example.txt" would be computed. This string is "ad0ca584e283ec89d67bd0ae23f14d1681ec1dd98db6c623e0820455c514dbef".

A to A- or from B+ to B) for my first offense, and that I will receive a grade of **"F" for the course** for any additional offense of any kind."

Compress the directory (e.g., `tar czvf jdoe-p1.tgz jdoe-p1`) and submit the tarball (in `tar.gz` or `tgz` format) to myCourses. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-p1.tar.gz` or `jdoe-p1.tgz`. Failure to use the right naming scheme will result in a 5% point deduction.

Your assignment will be graded on the CS Department computers `remote.cs.binghamton.edu`. It is your responsibility to make sure that your code compiles and runs correctly on CS Department computers. If you use external libraries in your code, you should also include them in your directory and correctly set the environment variables using absolute path in the Makefile. If your code does not compile on or cannot correctly run on the CS computers, you will receive no points.

References

[1] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.