Programming Assignment 2: Chord Distributed Hash Table

Due: October 29th, 2020 23:59pm

In this assignment, you will implement the basic functions of the Chord distributed hash table (DHT) [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++, or Java**.

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

1 Compile the interface definition file

I have compiled and installed Thrift on CS Department computers. These computers are located in G7, or you can also remotely access them by SSH'ing into `remote.cs.binghamton.edu`. To use the Thrift compiler, you need to set the environment variable PATH:

```
$> bash
$> export PATH=$PATH:/home/cs557-inst/local/bin
```

I have provided an Apache Thrift interface definition file, `chord.thrift`, for this assignment. You can locate this file on remote.cs computers at: `/home/cs557-inst/pa2/chord.thrift`. You can use Thrift to compile a `.thrift` interface definition file to Python, Java, or C++. As an example, the Thrift command to compile this assignment's IDL file to Java is as follows:

```
$> cp /home/cs557-inst/pa2/chord.thrift ./
$> 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 prepared 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/BingU-CS457-CS557/thrift-tutorial` . You should read the code in the language of your choice (one of Python, C++, and Java) 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 filename and contents, the corresponding file should be written to the server. Meta-information, including the filename and version number 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, i.e., the server is not the file's successor, a SystemException should be thrown.

As the set of servers on the Chord ring are stable in our case, you can run `findPred` (described beow) once to find the predecessor node of the server and store the returned information. This information can be used for checking if a server "owns" an id in all subsequent writeFile/readFile requests.

readFile if a file with a given name 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, i.e., the server is not the file's successor, 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, return 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 return 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, i.e., the first entry in the node's fingertable. A SystemException should be thrown if no fingertable exists for the current node.

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 filename.

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 `128.226.114.201`.

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 "sample.txt", then the key associated with this file SHA256("sample.txt")="9496..." 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...".

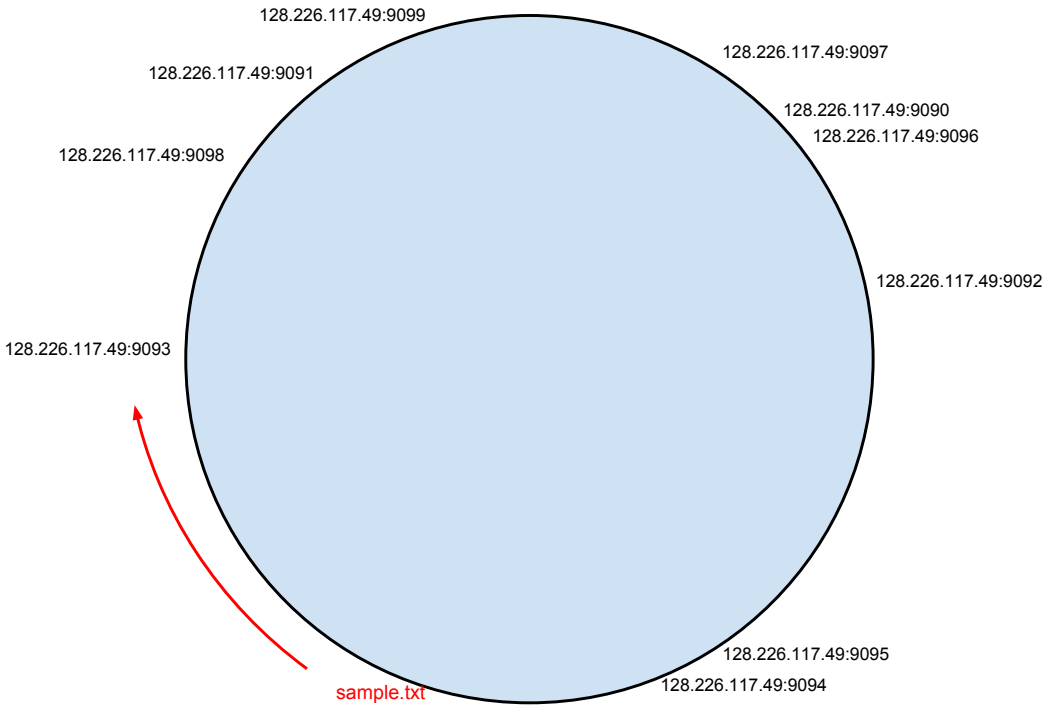

Figure 1: Example Chord Ring

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 create a single-threaded server using **TSimpleServer** and use Thrift's **TBinaryProtocol** for marshalling and unmarshalling data.

Multiple servers will be running at the same time as different Chord DHT nodes. Servers will 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 fingertable initializer program that will compute the fingertable for each running DHT node and send each fingertable (using the `setFingertable` server method you implement) to the appropriate node. The initializer program takes a filename as its only command line argument. To run the initializer:

```
$> /home/cs557-inst/pa2/init nodes.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:

```
128.226.114.201:9090
128.226.114.201:9091
128.226.114.201:9092
128.226.114.201:9093
```

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. That is, the server is not the successer of the file on the Chord ring. Your test program 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 filename: SHA-256("filename")[1] using the `findSucc` call. Then your test program can call `readFile` or `writeFile` on the server returned by the `findSucc` call.

To test the correctness of your `findPred` implementation, your test program can issue a `findPred` call to find the predecessor of a key whose id is the same as one server. It should expect the call to return the server that precedes the ID in counter-clockwise direction.

5 Github classroom

To access this assignment, first log into your Github account created with your BU email. Then go to: `https://classroom.github.com/a/bcSoUCZX`. After clicking this link, Github classroom will automatically create a repository. This is the repository you will push your code to.

This repository is a private repository. Only you, course instructor, and the course grader are able to see this repository. To clone this repository to your local file system,

```
git clone https://github.com/BingU-CS457-CS557/cs457-557-fall2020-pa2-username.git
```

Note that: i) `git` is already installed on CS department computers. To use it on your own computer, you must install it first; ii) you must replace "username" with your own Github username.

To add a file to the next commit, use the `git add` command. To commit your code, use the `git commit` command. Be sure to also push your commit to the Github repository after every commit using the `git push` command.

We will look into commit history to see if the code went through proper stages of software development. We expect each repository to have **at least five commits**, with the first one and the last one **more than 72 hours apart**. We expect to see at least one initial commit followed by commits that improve the code, e.g., fix bugs.

[1] SHA-256("filename") indicates computing the hexadecimal string associated with the SHA-256 hash of the filename, without quotes. For instance, if the filename is "sample.txt", then the 64-character SHA-256 string is "9496eec54d06963f3666d7719cd27073898a3ee588453b934627bb504cf19fbd".

6 How to submit

To submit, make a final commit with the message "final commit", and push your local repository to the private Github repository Github classroom created.

Your final commit should contain the following files:

1. Your source code which includes at least one file implementing the server.
2. 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.) If you used Python, then the Makefile is not required.
3. A `Readme` file describing:
   - the programming language(s) and tools (e.g., Apache Ant) that you used,
   - how to compile and run your code on `remote.cs.binghamton.edu` computers,
   - completion status of the assignment, e.g., what has been implemented and tested, what has not,
   - anything else you want the TA to be aware of while grading your assignment.
4. 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 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."

After pushing your final commit to the Github repository, please let us know by **submitting your commit hash to myCourses**.

After pushing your final commit to the Github repository, please let us know by **submitting your commit hash to myCourses**. The commit hash will be 40 characters long, e.g., "77469e1dada9586374520e34f0819ea246b6240b". There are two ways to locate your commit hash: You can type `git log` in your local repository, which will output the commit history. Locate the commit you want to use as your final submission, record the hash associated with the commit. Alternatively, you can also go to the Github page of your repository and locate it on the webpage.

It is important that you submit your commit hash to myCourses. This helps us know your submission is ready for grading and which of your commits we should grade. We will not grade your assignment unless you have submitted the commit hash to myCourses before the assignment submission deadline

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.

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.