Project 2: Basic Distributed Hash Table

Due: October 16th, 2014 23:59:59pm

In this project, you will create a basic distributed hash table (DHT) with an architecture similar to the Chord system [1]. The DHT will be built on the Apache Thrift Service from Project 1. Put and get hash table functions will be implemented as the writeFile and readFile functions of Project 1's service. Additional functions have been added to the interface to support a limited set of Chord DHT operations.

I have provided a file, `chord_dht.thrift`, that defines several of these operations.

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

1 Compile the interface definition file

As in Project 1, I have provided an Apache Thrift interface definition file, `chord_dht.thrift`. It can be downloaded from Blackboard. You can use Thrift to compile interface definition file to either Java, C++, or Python.

2 Extend the server-side method stubs generated by Thrift

You must implement methods corresponding to the following 7 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, creation-time, update-time, version, content length, 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.

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.

deleteFile if a file with a given name and owner exists on the server, it should be deleted, but the metadata should be retained and the deleted timestamp modified to record the time of deletion. If an error occurs during file deletion, e.g., deleting a non-existent file, 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 reference materials (Chapter 5.2.3 in the textbook and 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 project, 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 `write`, `read`, or `delete` a file 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 16 `REMOTE` machines ({remote00, remote01, ..., remote09, remote10, ..., remote15}.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...".

As in Project 1, 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. Unlike Project 1, 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 project, 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 Blackboard. The initializer program takes a filename as its only command line argument. To run the initializer:

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

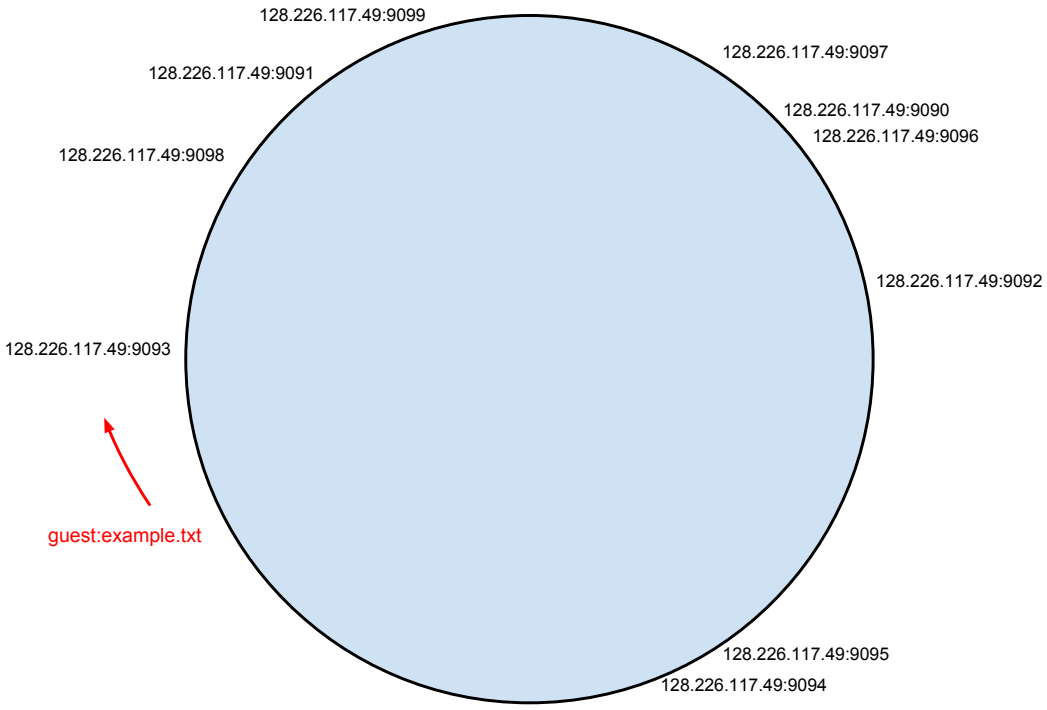

Figure 1: Example Chord Ring

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 Write a client program

You must also write a client program to `write`, `read`, and `delete` files from the DHT. As in Project 1, the client program should take five command line arguments as input. For example:

```
./client 128.226.180.163 9091 --operation write --filename example.txt --user guest
```

The first two arguments should be the host and port of one of the DHT nodes (should be an arbitrary node, not necessarily the node that actually stores the file). The final three arguments (occurring in no particular order) are as follows:

--operation [operation], where operation can be "read", "write", or "delete", and should invoke the respective remote procedure call **on the correct DHT node**.

--filename [filename], should be used as input to specify either a local file to be written to the remote system, or a remote filename to be read or deleted.

--user [user], should specify the owner of the file for the current operation.

Also as in Project 1, after the client program receives the response from a `read`, `write`, or `delete` remote procedure call, it should use a separate Thrift transport and protocol to write the returned structure to the standard output using Thift's TJSONProtocol. **The client should write any exceptions thrown by the remote method to the standard output using Thift's TJSONProtocol, but no output should be written for a `void` response.**

Unlike Project 1, the client must perform **more than one** RPC call for the `read`, `write`, or `delete` methods to operate correctly. The first RPC call should find the node that controls the key associated with the input user and filename: SHA-256("<user>:<filename>")[1] using the `findSucc` call. The next RPC call should call the `read`, `write`, or `delete` RPC on the node returned by the `findSucc` call.

5 How to submit

To submit the project, you should first create a directory whose name is "your BU email ID"`-project2`. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-project2`. You should put the following files into this directory:

1. The `chord_dht.thrift` file I provided.
2. The `init` initializer program I provided.
3. Your source code which includes at least two files implementing the client and the server, respectively.
4. A `Makefile` to compile your source code into two executables, which should be named `client` and `server`. (It is okay if these executables are bash scripts that call the Java interpreter, as long as the command line arguments follow the format described in Parts 2 & 4.)
5. A `Readme` file describing the programming language(s)/tools (e.g., Apache Ant) that you used and the implementation details of your program.

Compress the directory (e.g., `tar czvf jdoe-project2.tgz jdoe-project2`) and submit the tarball (in `tar.gz` or `tgz` format) to Blackboard. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-project2.tar.gz` or `jdoe-project2.tgz`. Failure to use the right naming scheme will result in a 5% point deduction.

Your project 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.

[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".