CSCI 5105: Introduction to Distributed Systems

Spring 2016

Instructor: Abhishek Chandra

Programming Assignment 1: Simple File System on a DHT

(*Due: Mar/2/2016 - midnight)*

1. Overview

In this programming assignment, you will implement a *simple file system* using a distributed hash table (DHT) based on the ***Chord*** protocol. Using this system, the client can write and read files using the DHT.

2. Project Details

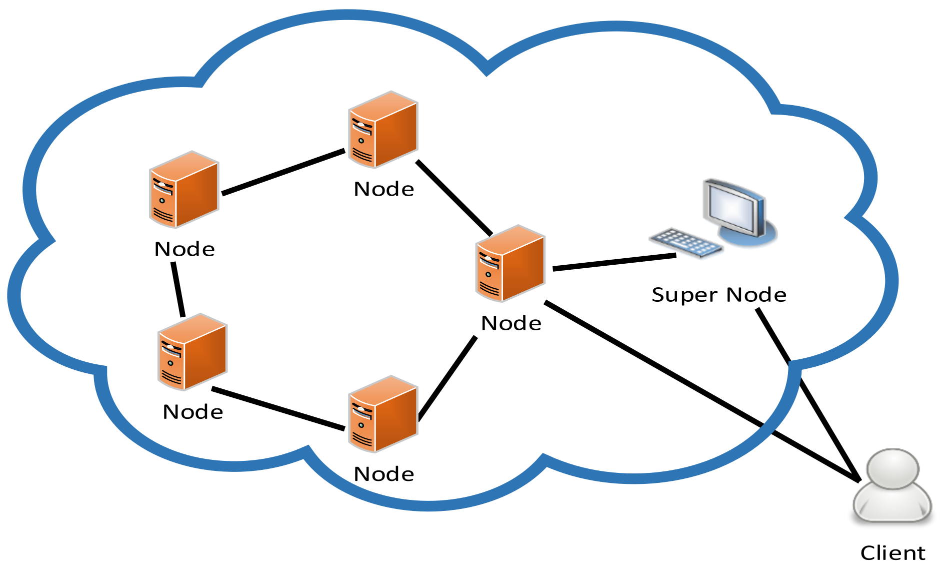
The operations which the client should be able to perform are:

a. Write files in the file system running on DHT. The file name should be used as the key into the DHT.
b. Read files. If there is no corresponding filename on DHT, the client will see appropriate error message.

The central components of the DHT are the nodes. They share the distributed store and its contents. In order to simplify your design the bootstrapping process for the DHT, you need to implement a SuperNode (a well-known node) for the DHT. The SuperNode will be running out of DHT but will help nodes configure DHT. That is, the SuperNode is the initial point of contact for the nodes to join the DHT and can be used to initiate the building of the DHT. The client also needs to contact to the SuperNode to get node information to contact.

In short, you will need to implement Client, Node, and SuperNode in this project.

a. **Client**: The client will be responsible for writing files to the file system as well as reading files from the file system.
b. **Nodes**: The nodes form the DHT with help from the SuperNode and are used by the client to perform “write” and “read” files. So nodes need to store files locally. In this project, <u>your DHT should have at least 5 nodes</u>.
c. **SuperNode**: This will help nodes form the DHT and let the client know node information to connect for sending requests. That is, the SuperNode needs to store all nodes information in the DHT.



Your underlying DHT is based on ***Chord***. **<u>It is highly recommended</u>** that you read the paper describing ***Chord*** to get an idea of how your system should behave.
The project will consist of following major steps:

a. Building the DHT: All the participating nodes will join the DHT with the help from the SuperNode.
b. Writing files to the DHT: A sample file list will be available with the client. Files will be partitioned across all the DHT nodes.
c. Reading files: Given any file name, the client will perform a “read” operation from the DHT to get its contents.

2.1. Building the DHT

The system starts off by initiating the DHT. You may need to insure that only one node is added to the DHT at a time for simplicity (to avoid any concurrency issue). For building the system, you need to implement the Chord “***Join***” protocol. Each node which wants to join the system, will contact the SuperNode which will give a list of nodes (including IP, Port, and Node ID) already joined. The node just joined to the DHT will (1) **parse** the list of nodes, (2) **set** its own node ID found from the list, (3) **build** a finger table, and (4) **distribute** the list to other nodes (found from the list) to let them update the finger table.

Follow list is an example of the list of nodes returned by the SuperNode.

- 127.0.0.1:54454:12,127.0.0.1:52212:21,127.0.0.1:52132:3

In this example, node information includes IP, port and ID (assigned by SuperNode) separated by colon. Each node information is separated by comma (You can use your own format if you want).

To ensure that only one node is joining the DHT at a time each node should inform the SuperNode when it has finished joining (and distributing the list of nodes). ID collisions can be avoided by having the SuperNode generate (and verify) the node IDs.

It would be a good idea to identify nodes by a host name (or IP address) and port number pair. Each node should maintain a finger table. The DHT should have some means of printing out its structure. This should include information about each node, such as its ID, its links (successor, predecessor), its finger table, and some information about the data that it is storing (e.g., the number of files) like below as an example.

- Node ID - range of keys - predecessor - successor - number of files stored
  File list
  Finger Table

It would be handy if the User Interface could print out this information. Note that the calculation of the finger tables and establishing forward links (successor, predecessor) is the responsibility of the individual nodes.

2.2. Writing files in the DHT

The client contains a list of files (files will be provided). The client is responsible for writing files into the DHT. Initially, the client will contact SuperNode and get a node (randomly chosen by SuperNode) to send operations. If the client gets a node information properly that is, DHT is ready to be used, it will simply send a “write” request to the node.

The file names are hashed by the node to determine the node ID which is responsible for that filename. You may use any reasonable hash function (e.g., md5 hash) for hashing the filename.

If the node is not responsible for the filename, the node forwards the request recursively. That is, the node will forward (route) the request to other nodes according to the DHT table. The forwarding will be done synchronously until the request is handled by a node responsible for the filename. <u>You need to track the nodes visited to handle the request</u>.

2.3. Reading files from the DHT

The client also perform a “read” operation where it takes input as a filename (key). This operation will be similar to the “write” operation, where the client will contact the SuperNode for getting a node ID. It will then contact that node with the read request. If the node does not contain the file, (1) it will forward (route) the request to other nodes according to the finger table recursively. If the file is not present in DHT, appropriate error message should be displayed.

3. Implementation Details

Your system may be implemented in C++ or Java using ***Thrift***.

The SuperNode will contain an interface for the client and nodes. Following calls to the SuperNode must be implemented:

1. `Join(IP, Port)`**:** When a node wants to join the DHT, it contacts the SuperNode using Thrift. The SuperNode will then return the list of nodes. If the SuperNode is busy in join process of another node, it will return a “NACK” to the requesting node.
2. `PostJoin(IP, Port)`**:** After the node is done to join the DHT (also done to distribute the list of nodes to other nodes), it should notify the SuperNode about it. Only after getting this “Done” message, the SuperNode can allow other nodes to join the DHT. This will prevent nodes from getting added concurrently.
3. `GetNode()`**:** For sending requests to DHT, the client should know the node information. For this, the client will contact to the SuperNode and it will return Node information randomly chosen. The client can contact SuperNode only once when the client is running or every time when the client sends a request for testing purpose.

The Node will contain an interface for the client and other nodes. Following calls to the Node must be implemented:

1. `Write(Filename, Contents)`**:** When a client wants to write a file, it contacts the Node using Thrift. The Node will check whether it needs to store the file locally or not. If it is not the node for the filename, it will forward the request to other nodes ***recursively***.
2. `Read(Filename)`**:** When a client wants to read a file, it contacts the Node using Thrift. The Node will check whether it is the node for the filename. If not, it will forward the request to other nodes ***recursively***.
3. `UpdateDHT(NodesList)`**:** When a node joined to DHT, it will contact to nodes in the list one-by-one to let them update DHT with the new nodes list. When the node finishes calling to all nodes, it will let the SuperNode know that it is done to join by calling `PostJoin()`.

You can modify these interfaces and add any other interfaces if needed. The “Read” and “Write” operations should also have an option of printing out log messages when required. For example: During the “Read” operation, the system should print out all the list of nodes visited, the node where the file was stored etc. For this, you can use any data format (JSON, XML) if it is needed.

Assumptions and Hints:

- Each Node can either run on same or different machine on its own port.
- More than 2 Thrift Interface files are needed (for SuperNode and Nodes).
- The Nodes will act as client of SuperNode for joining phase for forming DHT and will act a server for handling requests from the client.
- SuperNode should not maintain any state about the DHT (only the list of nodes)
- For simplicity, the file contents will be very simple. E.g., happy.txt stores “happy”. The client can provide UI for testing write and read operations. E.g., the client can input the filename and contents in UI for write a file.
- The file system does not need to be persistent in this project.
- The number of nodes for DHT can be set when the SuperNode starts as a parameter. (This will let the SuperNode know the DHT is ready)
- To simplify your design, you need not worry about node failures or nodes leaving the DHT after they've joined.
- For the forwarding (routing), you will need to consider how to avoid infinite loop.

Note: You should be building a distributed, peer-to-peer system where the SuperNode provides just a little bit of help to build DTH and to avoid synchronization. You should not be building a centralized system where most of the intelligence is in the SuperNode while the other nodes are relatively dumb. The SuperNode is just a supervisor node and does not store actual key-value pair information but only nodes list.

4. Project Group

All students should work in groups of size no more than 2 members.

5. Testcases

Basic test cases include a positive test case – where the filename is present in the DHT and a negative test case – where the file is not present in the DHT.

You should also develop your own test cases for all the components of the architecture, and provide documentation that explains how to run each of your test cases, including the expected results. Be creative, and do something fun!

Note that all these services are expected to work normally when deployed across different machines as well as when deployed over a single machine.

6. Deliverables

- Design document describing each component.
- User document explaining how to run each component and how to use the service as a whole, including command line syntax, configuration file particulars, and user input interface
- Testing description, including a list of cases attempted (which must include negative cases) and the results.
- Source code, Makefiles and/or a script to start the system. (Not any object file)

7. Grading

The grade for this assignment will include the following components:

- 20% - The document you submit
  - Detailed description of the system design and operation
  - Test cases used for the system (must include negative cases)
- 70% - The functionality and correctness of your program
  - Forming (Updating) DHT – 30%
  - Read, Write operations
    - Recursive operation – 30%
  - Printing Node information. – 5%
  - Tracking requests (Routing info). – 5%
- 10% - The quality of the source code, in terms of style and in line documentation

If DHT is not formed properly, you will not get full credit for other functionalities. You will lose points if there is any exception, crash or freezing on your programs.

8. References

- I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In Proc. ACM SIGCOMM’01, San Diego, CA, Aug. 2001.
- Thrift White paper

- https://thrift.apache.org/static/files/thrift-20070401.pdf

- How to setup Thrift in your own machine (Ubuntu)
  - Packages for compiling Thrift
    https://thrift.apache.org/docs/install/debian
  - Building Thrift from source codes
    https://thrift.apache.org/docs/BuildingFromSource
- Tutorial by Examples
  - Java
    https://thrift.apache.org/tutorial/java
    http://thrift-tutorial.readthedocs.org/en/latest/usage-example.html
  - C/C++
    https://thrift.apache.org/tutorial/cpp