CSCI 5105: Introduction to Distributed Systems

Spring 2019

Instructor: Abhishek Chandra

Programming Assignment 2: Book Finder System on a DHT

(Due: Apr/3/2019 – 10pm)

1. Overview

In this programming assignment, you will implement a *Book Finder System* using a distributed hash table (DHT) based on the ***Chord*** protocol. Using this system, the client can set and get book's information (title and genre for simplicity) using the DHT.

2. Project Details

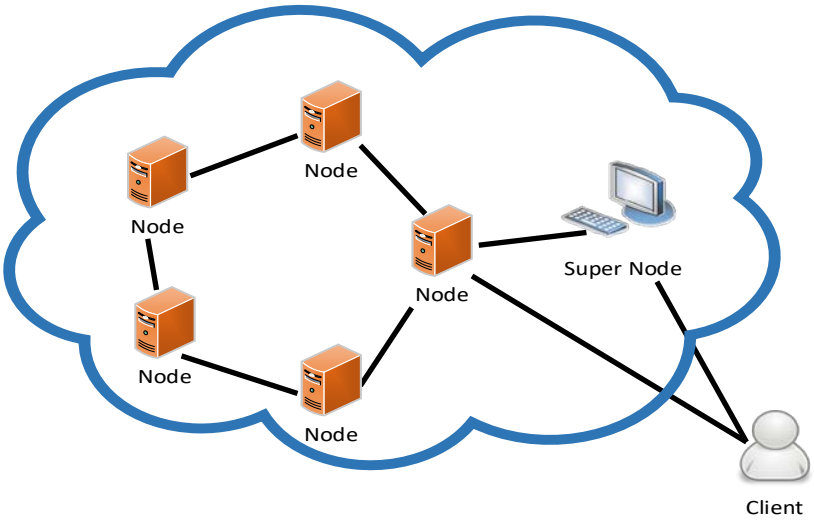
The operations which the client should be able to perform are:

a. Setting a book title (e.g., All's Well That Ends Well) and its genre (e.g., Comedies) in the system running on DHT. The book title should be used as the key into the DHT.
b. Getting a genre with a book title. If there is no book title 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 and the bootstrapping process for the DHT, you need to implement a SuperNode (an external well-known node) for the DHT. The SuperNode will be running outside the 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 setting book titles and genres to the system as well as getting a genre from the system with a book title.
b. **Nodes**: The nodes form the DHT with help from the SuperNode and are used by the client to perform "set" and "get" book information. Nodes need to store book titles and genres locally. In this project, your DHT should have at least 5 nodes.
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***. **It is highly recommended** 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. Setting book titles and genres to the DHT: A sample file will be available with the client. The book information will be partitioned across all the DHT nodes based on book titles.
c. Getting a genre: Given any book title, the client will perform a "get" operation from the DHT to get its genre.

2.1. Building the DHT

The system starts off by initiating the DHT. You may need to ensure 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 assign a node ID to the node and provide information about an existing DHT node (including IP and Port). Given information, the node joining the DHT will (1) **find** a successor and a predecessor (2) **build** a finger table, and (3) **distribute** update notice to predecessors to let them update their finger tables. You can find detail of the algorithms in the ***Chord*** paper (Figures 4 and 6).

Following is an example of node information returned by the SuperNode for a new node.

- 5,127.0.0.1:54454:12

In this example, 5 is the node ID assigned by the SuperNode for the new node. The following (after comma) string is node information of one of nodes that includes IP, port and ID separated by colon. The new node will connect to this node to join the DHT. 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 cities) like below as an example.

- Node ID - range of keys - predecessor - successor - number of cities stored
  Cities 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. Setting Book-Genre in the DHT

The client sets a list of book title and genre pair (list of pairs will be provided). The client is responsible for setting this information into the DHT. Initially, the client will contact the SuperNode and get a node (randomly chosen) to send operations. If the client gets node information properly i.e., DHT is ready to be used, the client will simply send a “set” request to the node. The client needs to provide a way to set many book title and genre pairs in a given file at once. You will need to implement multi-threaded server for a node as there can be multiple clients in the system at the same time.

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

If the node is not responsible for the book title, 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 book title. You need to track the nodes visited to handle the request.

2.3. Getting Genre from the DHT

The client gets a genre with a book title. This operation will be like the “set” operation, where the client will contact the SuperNode for getting a node ID. It will then contact that node with the get request. The book title is hashed by the node to determine the node ID which is responsible for that book title. If the node does not contain the book title, (1) it will forward (route) the request to other nodes according to the finger table recursively. If the book title is not present in DHT, appropriate error message should be displayed.

3. Implementation Details

Your system will 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 one of nodes information. With given information, the new node will build the DHT. 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 may 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. `Set(Book_title, Genre):` When a client wants to set a book title and a genre, it contacts the Node using Thrift. The node will check whether it needs to store information locally or not. If it is not the node for the book title, it will forward the request to other nodes ***recursively***.
2. `Get(Book_title):` When a client wants to know a genre with a book title, it contacts the Node using Thrift. The Node will check whether it is the node for the book title. If not, it will forward the request to other nodes ***recursively***.
3. `UpdateDHT():` After joining to DHT, the new node will contact to nodes in the finger table to let them update DHT. 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 “Set” and “Get” operations should also have an option of printing out log messages when required. For example: During the “Get” operation, the system should print out all the list of nodes visited, the node where the book title was stored etc.

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)**
- The client can provide UI for testing Set and Get operations,
  e.g., the client can input a book title and a genre in UI, and the client can set all cities from a file.
- The genre will be updated if a client sets a different genre for a book title.
- The 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 to 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 book title is present in the DHT and a negative test case – where the book title 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.
- **Please do not include any ".git", ".idea", "_MACOSX" directories, classe (object) files, and given dataset in your submission.**
- Only documents, source code, Makefiles and/or a script to start the system need to be included in your submission.

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%
  - Put, Get 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