CSE 486/586 Distributed Systems Programming Assignment 3

Simple DHT

Introduction

In this assignment, you will design a simple DHT based on Chord. Although the design is based on Chord, it is a simplified version of Chord; you do not need to implement finger tables and finger-based routing; you also do not need to handle node leaves/failures.Therefore, there are three things you need to implement: 1) ID space partitioning/re-partitioning, 2) Ring-based routing, and 3) Node joins.

Just like the previous assignment, your app should have an activity and a content provider. However, the main activity should be used for testing only and should not implement any DHT functionality. The content provider should implement all DHT functionalities and support insert and query operations. Thus, if you run multiple instances of your app, all content provider instances should form a Chord ring and serve insert/query requests in a distributed fashion according to the Chord protocol.

References

Before we discuss the requirements of this assignment, here are two references for the Chord design:

1. Lecture slides on Chord
2. Chord paper

The lecture slides give an overview, but do not discuss Chord in detail, so it should be a good reference to get an overall idea. The paper presents pseudo code for implementing Chord, so it should be a good reference for actual implementation.

Note

It is important to remember that this assignment does not require you to implement everything about Chord. Mainly, there are three things you ***do not*** need to consider from the Chord paper.

1. Fingers and finger-based routing (i.e., Section 4.3 & any discussion about fingers in Section 4.4)
2. Concurrent node joins (i.e., Section 5)
3. Node leaves/failures (i.e., Section 5)

We will discuss this more in “Step 2: Writing a Content Provider” below.

Step 0: Importing the project template

Just like the previous assignment, we have a project template you can import to Android Studio.

1. Download the project template zip file to a directory.
2. Extract the template zip file and copy it to your Android Studio projects directory.

   a. Make sure that you copy the correct directory. After unzipping, the directory name should be “SimpleDht”, and right underneath, it should contain a number of directories and files such as “app”, “build”, “gradle”, “build.gradle”, “gradlew”, etc.
3. After copying, delete the downloaded template zip file and unzipped directories and files. This is to make sure that you do not submit the template you just downloaded. (There were many people who did this before.)
4. Open the project template you just copied in Android Studio.
5. Use the project template for implementing all the components for this assignment.
6. The template has the package name of “edu.buffalo.cse.cse486586.simpledht“. Please do not change this.
7. The template also defines a content provider authority and class. Please use it to implement your Chord functionalities.
8. We will use SHA-1 as our hash function to generate keys. The following code snippet takes a string and generates a SHA-1 hash as a hexadecimal string. Please use it to generate your keys. The template already has the code, so you just need to use it at appropriate places. Given two keys, you can use the standard lexicographical string comparison to determine which one is greater.

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Formatter;

private String genHash(String input) throws NoSuchAlgorithmException {
        MessageDigest sha1 = MessageDigest.getInstance("SHA-1");
        byte[] sha1Hash = sha1.digest(input.getBytes());
        Formatter formatter = new Formatter();
        for (byte b : sha1Hash) {
                formatter.format("%02x", b);
        }
        return formatter.toString();
}
```

Step 1: Writing the Content Provider

First of all, your app should have a content provider. This content provider should implement all DHT functionalities. For example, it should create server and client threads (if this is what you decide to implement), open sockets, and respond to incoming requests; it should also implement a simplified version of the Chord routing protocol; lastly, it should also handle node joins. The following are the requirements for your content provider:

1. We will test your app with any number of instances up to 5 instances.
2. The content provider should implement all DHT functionalities. This includes all communication as well as mechanisms to handle insert/query requests and node joins.
3. Each content provider instance should have a node id derived from its emulator port.

This node id should be obtained by applying the above hash function (i.e., genHash()) to the emulator port. For example, the node id of the content provider instance running on emulator-5554 should be, *node_id = genHash("5554")*. This is necessary to find the correct position of each node in the Chord ring.

4. Your content provider should implement insert(), query(), and delete(). The basic interface definition is the same as the previous assignment, which allows a client app to insert arbitrary <"key", "value"> pairs where both the key and the value are strings.
    a. For delete(URI uri, String selection, String[] selectionArgs), you only need to use use the first two parameters, uri & selection. This is similar to what you need to do with query().
    b. *However, please keep in mind that this "key" should be hashed by the above genHash() before getting inserted to your DHT in order to find the correct position in the Chord ring.*
5. For your query() and delete(), you need to recognize two special strings for the *selection* parameter.
    a. If * (not including quotes, i.e., "*" should be the string in your code) is given as the *selection* parameter to query(), then you need to return all <key, value> pairs stored in your entire DHT.
    b. Similarly, if * is given as the *selection* parameter to delete(), then you need to delete all <key, value> pairs stored in your entire DHT.
    c. If @ (not including quotes, i.e., "@" should be the string in your code) is given as the *selection* parameter to query() on an AVD, then you need to return all <key, value> pairs *stored in your local partition of the node*, i.e., all <key, value> pairs stored locally in the AVD on which you run query().
    d. Similarly, if @ is given as the *selection* parameter to delete() on an AVD, then you need to delete all <key, value> pairs *stored in your local partition of the node*, i.e., all <key, value> pairs stored locally in the AVD on which you run delete().
6. An app that uses your content provider can give arbitrary <key, value> pairs, e.g., <"I want to", "store this">; then your content provider should hash the key via genHash(), e.g., genHash("I want to"), get the correct position in the Chord ring based on the hash value, and store <"I want to", "store this"> in the appropriate node.
7. Your content provider should implement ring-based routing. Following the design of Chord, your content provider should maintain predecessor and successor pointers and forward each request to its successor until the request arrives at the correct node. Once the correct node receives the request, it should process it and return the result (directly or recursively) to the original content provider instance that first received the request.
    a. Your content provider do not need to maintain finger tables and implement finger-based routing. This is not required.
    b. As with the previous assignment, we will fix all the port numbers (see below). This means that you can use the port numbers (11108, 11112, 11116, 11120, & 11124) as your successor and predecessor pointers.
8. Your content provider should handle new node joins. For this, you need to have the first emulator instance (i.e., emulator-5554) receive all new node join requests. Your

implementation should not choose a random node to do that. Upon completing a new node join request, affected nodes should have updated their predecessor and successor pointers correctly.

   a. Your content provider do not need to handle concurrent node joins. You can assume that a node join will only happen once the system completely processes the previous join.
   b. Your content provider do not need to handle insert/query requests while a node is joining. You can assume that insert/query requests will be issued only with a stable system.
   c. Your content provider do not need to handle node leaves/failures. This is not required.

9. We have fixed the ports & sockets.
   a. Your app should open one server socket that listens on 10000.
   b. You need to use run_avd.py and set_redir.py to set up the testing environment.
   c. The grading will use 5 AVDs. The redirection ports are 11108, 11112, 11116, 11120, and 11124.
   d. You should just hard-code the above 5 ports and use them to set up connections.
   e. Please use the code snippet provided in PA1 on how to determine your local AVD.
      i. emulator-5554: “5554”
      ii. emulator-5556: “5556”
      iii. emulator-5558: “5558”
      iv. emulator-5560: “5560”
      v. emulator-5562: “5562”

10. Your content provider’s URI should be: “content://edu.buffalo.cse.cse486586.simpledht.provider”, which means that any app should be able to access your content provider using that URI. Your content provider does not need to match/support any other URI pattern.

11. As with the previous assignment, Your provider should have two columns.
   a. The first column should be named as “key” (an all lowercase string without the quotation marks). This column is used to store all keys.
   b. The second column should be named as “value” (an all lowercase string without the quotation marks). This column is used to store all values.
   c. All keys and values that your provider stores should use the string data type.

12. Note that your content provider should only store the <key, value> pairs local to its own partition.

13. Any app (not just your app) should be able to access (read and write) your content provider. As with the previous assignment, please do not include any permission to access your content provider.

14. Please read the notes at the end of this document. You might run into certain problems, and the notes might give you some ideas about a couple of potential problems.

Step 2: Writing the Main Activity

The template has an activity used for *your own testing and debugging.* It has three buttons, one button that displays "Test", one button that displays "LDump" and another button that displays "GDump." As with the previous assignment, "Test" button is already implemented (it's the same as "PTest" from the last assignment). You can implement the other two buttons to further test your DHT.

1. LDump
    a. When touched, this button should dump and display all the <key, value> pairs *stored in your local partition of the node*.
    b. This means that this button can give @ as the selection parameter to query().
2. GDump
    a. When touched, this button should dump and display all the <key, value> pairs stored in your ***whole*** DHT. Thus, LDump button is for local dump, and this button (GDump) is for global dump of the entire <key, value> pairs.
    b. This means that this button can give * as the selection parameter to query().

Testing

We have testing programs to help you see how your code does with our grading criteria. If you find any rough edge with the testing programs, please report it on Piazza so the teaching staff can fix it. The instructions are the following:

1. Download a testing program for your platform. If your platform does not run it, please report it on Piazza.
    a. Windows: We've tested it on 32- and 64-bit Windows 8.
    b. Linux: We've tested it on 32- and 64-bit Ubuntu 12.04.
    c. OS X: We've tested it on 32- and 64-bit OS X 10.9 Mavericks.
2. Before you run the program, please make sure that you are running five AVDs. `python run_avd.py 5` will do it.
3. Run the testing program from the command line.
4. On your terminal, it will give you your partial and final score, and in some cases, problems that the testing program finds.

Submission

We use the CSE submit script. You need to use either "submit_cse486" or "submit_cse586", depending on your registration status. If you haven't used it, the instructions on how to use it is here: https://wiki.cse.buffalo.edu/services/content/submit-script

You need to submit one file described below. Once again, you must follow everything below exactly. Otherwise, you will get no point on this assignment.

- Your entire Android Studio project source code tree zipped up in .zip: The name should be SimpleDht.zip.

a. Never create your zip file from inside “SimpleDht” directory.
b. Instead, make sure to zip “SimpleDht” directory itself. This means that you need to go to the directory that contains “SimpleDht” directory and zip it from there.
c. Please do not use any other compression tool other than zip, i.e., no 7-Zip, no RAR, etc.

Deadline: **4/12/2017 (Friday) 11:59:59am**

The deadline is firm; if your timestamp is 12pm, it is a late submission.

Grading

This assignment is 10% of your final grade. The breakdown for this assignment is:

- 1% if local insert/query/delete operations work correctly with 1 AVD.
- Additional 2% if the insert operation works correctly with static/stable membership of 5 AVDs.
- Additional 2% if the query operation works correctly with static/stable membership of 5 AVDs.
- Additional 2% if the insert operation works correctly with 1 - 5 AVDs.
- Additional 2% if the query operation works correctly with 1 - 5 AVDs.
- Additional 1% if the delete operation works correctly with 1 - 5 AVDs.

Notes

- Please do not use a separate timer to handle failures. This will make debugging very difficult. Use socket timeouts and handle all possible exceptions that get thrown when there is a failure. They are:
    - SocketTimeoutException, StreamCorruptedException, IOException, FileException, and EOFException.
- Please use full duplex TCP for both sending and receiving. This means that there is no need to create a new connection every time you send a message. If you’re sending and receiving multiple messages from a remote AVD, then you can keep using the same socket. This makes it easier.
- Please do not use Java object serialization (i.e., implementing Serializable). It will create large objects that need to be sent and received. The message size overhead is unnecessarily large if you implement Serializable.
- Please do not assume that there is a fixed number of messages (e.g., 25 messages) sent in your system. Your implementation should not hardcode the number of messages in any way.
- There is a cap on the number of AsyncTasks that can run at the same time, even when you use THREAD_POOL_EXECUTOR. The limit is "roughly" 5. Thus, if you need to create more than 5 AsyncTasks (roughly, once again), then you will have to use something else like Thread. However, I really do not think that it is necessary to create that many AsyncTasks for the PAs in this course. Thus, if your code doesn't work because you hit the AsyncTask limit, then please think hard why you're creating that

many threads in the first place.

This document gives you more details on the limit and you might (or might not, depending on your background) understand why I say it's "roughly" 5.

http://developer.android.com/reference/java/util/concurrent/ThreadPoolExecutor.html
(Read "Core and maximum pool sizes.")

- For Windows users: In the past, it was discovered that sometimes you cannot run a grader and Android Studio at the same time. As far as I know, this happens rarely, but there is no guarantee that you will not encounter this issue. Thus, if you think that a grader is not running properly and you don't know why, first try closing Android Studio and run the grader.