CSE 486/586: Distributed Systems

Programming Assignment 3[1]
Simple DHT

Introduction

For this assignment you will design a simple DHT based on Chord. Although the design described here is based on Chord, it is a simplified version of Chord. You will not need to implement finger tables or finger-based routing, and you do not need to handle nodes leaving the DHT or failing.

Like the previous assignment, your app will have an activity and a content provider. However, the main activity should be used *only for testing* and should not implement *any DHT functionality*. The content provider should implement all DHT functionality and support insert and query functions on the DHT. Multiple content provider instances should form a Chord ring and serve insert and query requests in a distributed fashion according to the Chord protocol.

You may use code from your previous projects in this project. Please remember to cite any code that comes from sources such as the Oracle tutorials or Android Developers.

Please read this document in its entirety before you begin. It is long, but that is only because it is complicated and precise. Revisit the instructions regularly as you implement to be sure you're implementing what is required. There are notes at the end that may assist you if you encounter common problems; make sure to review them as you go, as well.

This assignment will require you to:

- Implement ID space partitioning and re-partitioning
- Implement ring-based routing
- Handle node joins to a DHT containing data

1 Getting Started

We are providing a project template for you to get started. As in the previous assignments, the template configures a number of things on your behalf. It is important that you do not change settings that you do not have to change to complete your project.

You will need to download the project from GitHub Classroom and open it in Android Studio. You should have received the GitHub Classroom invitation for this project via Piazza.

Instructions for cloning the project and opening it in Android Studio are the same as before, except that the base repository name is `SimpleDht`. You can find YouTube tutorials detailing the process here.

[1]This assignment is based, with permission, on an assignment developed and used by Steve Ko in his version of CSE 486/586. Large portions of text are used verbatim from that assignment.

2 The Content Provider

The graded portion of this project is a `ContentProvider` using DHT semantics similar to a simplified Chord to store and retrieve data across multiple devices. The provided template uses the package name `edu.buffalo.cse.cse486586.simpledht`, and defines a content provider authority and class. Please do not change the package name, and use the content provider authority and class for your implementation.

You will use SHA-1 as the hash function to generate IDs for the DHT. The following code snippet takes a Java String and uses it to generate a SHA-1 hash as a hexadecimal string. Use this to generate your keys. The project template already includes this code, you must simply use it in the appropriate places. Given two keys, you can use the standard lexicographical String comparison (*i.e.*, `String.compareTo(String)`) 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();
}
```

For this project, you will need to run five emulators. We will be using the multi-port configuration as described in project 1. Your app will open one server socket that listens on port 10000, but it will connect to a different port number on the IP address 10.0.2.2 for each emulator, as follows:

emulator serial	port
emulator-5554	11108
emulator-5556	11112
emulator-5558	11116
emulator-5560	11120
emulator-5562	11124

2.1 Implementing the Content Provider

Your application content provider should implement all of the required DHT functionalities. For example, it should create server and client threads (if you choose to use them in your implementation), open sockets, and respond to incoming requests. It should also implement the simplified version of the Chord routing protocol specified here. Finally, it should also handle node joins. The specific requirements for your content provider follow.

2.1.1 Requirements

1. The URI of your content provider must be:
`content://edu.buffalo.cse.cse486586.simpledht.provider`

2. The content provider should implement all DHT functionalities. This includes all communication as well as mechanisms to handle insert and query requests from other nodes and to handle node joins.

3. Each content provider instance should have a node ID derived from its emulator serial number. This node ID *must be obtained by applying the specified hash function (named* `genHash()`*, above) to the emulator serial number.* For example, the node ID of the content provider running on emulator-5554 should be `node_id = genHash("5554")`. The ID string you should use is exactly the string found in `portStr` in the telephony hack. This is necessary to place each node in the correct place in the Chord ring.

4. Your content provider must implement `insert()`, `query()`, and `delete()`. The basic interface definition is the same as in Project 2, parts A and B, which allows a client to manipulate arbitrary key-value pairs, where both the key and value are Java strings.

   - For `delete(URI uri, String selection, String selectionArgs)` you need only use the first two paramters, `uri` and `selection`, similar to our previous implementations of `query()`.
   - The key *must be hashed by* `genHash()` *before being inserted in the DHT in order to determine its position in the Chord ring.*

5. For both `query()` and `delete()`, you must recognize two special strings, `"*"` and `"@"`, for the `selection` parameter.

   - If * (not including quotes; `"*"` should be the literal string in your code) is given as the `selection` parameter to `query()`, then you must return *all key-value pairs stored in your entire DHT.*
   - Similarly, if * is given as the `selection` parameter to `delete()`, then you must delete *all key-value pairs stored in your entire DHT.*
   - If @ (again, not including quotes) is given as the `selection` parameter to `query()` on a given AVD, then you must return all key-value pairs *stored on the AVD running the query*, but not any other AVDs in the DHT.
   - Similarly, if @ is given as the `selection` parameter to `delete()` on an AVD, then you must delete all key-value pairs *stored on the AVD running the delete*, but not on any other AVDs in the DHT.

6. Any app using your content provider can submit arbitrary key-value pairs of Java strings (*e.g.*, <`"I want to"`, `"store this"`>). Your content provider must hash the key using `genHash()` (in this example, `genHash("I want to")`) to find the correct location in the Chord ring for this datum, then store <`"I want to"`, `"store this"`> on the appropriate node.

7. Your content provider must implement ring-based routing (but need not implement Chord finger routing or finger tables). Following the Chord ring design, your content provider should maintain predecessor and successor pointers, then forward each request for data not stored locally to its successor until the request arrives at the correct node. When a node receives a request for an ID that it maintains, it should process the request and send the result (either directly or recursively, at your discretion) to the content provider instance initiating the request.

8. Your content provider should handle new node joins. *For this functionality, the emulator instance emulator-5554 should receive all new node join requests.* You should not choose a random node to receive new node join requests, and you should start the content provider on emulator-5554 *first* to enable this. upon completing a new node join request, affected nodes should update their predecessor and successor pointers correctly.

    - Your content provider need not handle concurrent node joins. You can assume that a node join will only happen after the system has completely processed any previous node joins.
    - Your content provider need not handle any insert, query, or delete requests while a node is joining. You may assume that requests will be issued only when the system is stable.
    - Your content provider need not handle node leaves or failures.

9. As in the previous assignments, your content provider should have two columns.

    - The first column should be named "key" (all lowercase, no quotation marks). This column is used to store keys.
    - The second column should be named "value" (all lowercase, no quotation marks). This column is used to store the values associated with keys.
    - All keys and values stored by your provider should be Java strings.

10. Any app should be able to access (both read and write) your content provider. As in previous assignments, please do not require any permissions to access your content provider.

Note that each content provider must store only *the key-value pairs local to its own partition of the Chord ID space!*

2.2 Writing the main activity

The provided template code includes an activity to be used for your own testing and debugging. *It will not be specifically graded or evaluated in any way.* It has three buttons; one button that displays "Test", one button that displays "LDump", and another button that displays "GDump". As in the previous assignment, the "Test" button is already implemented (and it is the same as "PTest" in Programming Assignment 2 part B). You can (and should) implement the other two buttons to further test your DHT.

- *LDump*
  - When pressed, this button should dump and display all of the key-value pairs *stored on the local partition of the DHT*.
  - This can be accomplished by issuing a query with @ as the selection parameter, and printing the results.
- *GDump*
  - When pressed, this button should dump and display all of the key-value pairs *in the entire DHT*.
  - This can be accomplished by issuing a query with * as the selection parameter, and printing the results.
- In this manner, the LDump button will provide a "local dump" of the current AVD, and GDump will provide a "global dump" of the contents of the entire ring.

You may find it useful or desirable to implement additional functionality in your main activity, and that is fine.

2.3 Testing

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

- Download a testing program for your platform. If your platform does not run it, please report it.
  1. Windows: Tested on 64-bit Windows 8.
  2. Linux: Tested on 64-bit Debian 9 and 64-bit Ubuntu 17.10 (see below for important information about 64-bit systems).
  3. Mac OS: Tested on 64-bit Mac OS 10.9 Mavericks.
- Before you run the program, please make sure that you are running all five AVDs. You can use `python run_avd.py 5` to start them.
- Remember to start the emulator network by running `set_redir.py 10000`.
- Like the previous testing program, this test will require you to provide your APK as a command line argument, and will take care of installing and uninstalling it on the emulators.
- Run the testing program from the command line.
- It may issue some warnings or errors during execution. Some of them are normal, some may indicate errors in your program. Examine them to find out!

- The testing program will give you partial and final scores.

Remember during your own testing that you may see strange contents in your content provider if you do not uninstall your application between tests. In particular, updating the app from Android Studio does not uninstall the existing app first, so the content provider's state will not be cleared. You can uninstall your app by hand with the following command:

```
adb -s <emulator> uninstall edu.buffalo.cse.cse486586.simpledht
```

Notes for 64-bit Linux: The testing program is compiled 32-bit. If you get an error like the following, or the shell reports command not found when you run the executable, install the 32-bit `libz` for your system:

```
./simpledht-grading.linux: error while loading shared libraries:
    libz.so.1: cannot open shared object file: No such file or directory
```

On Debian-based distributions, you can accomplish this with the command `apt-get install zlib1g:i386` as root (you may need to use `sudo` or `su`). If `apt-get` reports an error about the architecture or says the package is not found, you may need to enable multiarch. To do this, run `dpkg --add-architecture i386` as root, then update your APT repositories with `apt-get update` as root. Once this is done, you should be able to install the 32-bit `libz`.

For other distributions you will need to consult your distribution documentation.

3 Submission

We use UB CSE autograder for submission. You can find autograder at https://autograder.cse.buffalo.edu/, and log in with your UBITName and password.

Once again, *it is critical that you follow everything below exactly.* Failure to do so **will lead to no credit for this assignment**.

Zip up your entire Android Studio project source tree in a single zip file. Ensure that *all* of the following are true:

1. You *did not* create your zip file from *inside* the `GroupMessenger1` directory.
2. The top-level directory in your zip file is `SimpleDht` or `SimpleDht-<something>`, and it contains `build.gradle` and all of your sources.
3. You used a zip utility and *not any other compression or archive tool:* this means no 7-Zip, no RAR, no tar, *etc.*

4 Deadline

This project is due 2018-04-13 11:59:00 AM. This is one hour before our class. This is a firm deadline. If the timestamp on your submission is 11:59:01, it is a late submission. You are expected to attend class on this day!

Note that *this deadline has been pre-extended, and it will not be extended any further.* Extending this deadline will not leave sufficient time to both complete Programming Assignment 4 and study for Finals.

5 Grading

This assignment is 10% of your final grade. Credit for this assignment will be apportioned as follows:

- 1%: Local insert/query/delete operations work on a DHT containing a single AVD.
- 2%: The insert operation works correctly with static membership of 5 AVDs.
- 2%: The query operation works correctly with static membership of 5 AVDs.
- 2%: The insert operation works correctly with between 1 and 5 AVDs (and possibly changing membership).
- 2%: The query operation works correctly with between 1 and 5 AVDs (and possibly changing membership).
- 1%: The delete operation works correctly with between 1 and 5 AVDs (and possibly changing membership).

Thus, an application providing all of the required features would receieve 10 total points, for 10% of the final course grade.

Notes

- Please do not use a separate timer to handle failures, as this will make debugging very difficult and is likely to introduce race conditions. Use socket timeouts and handle all possible exceptions that may be thrown when there is a failure. They are: `SocketTimeoutException`, `StreamCorruptedException`, `EOFException`, and the parent `IOException`.
- Please reuse your TCP connections with a single remote host, instead of creating a new socket every time you send a message. You can use the same socket for both sending to and receiving from a remote AVD. Think about how you will coordinate this.
- Please do not use Java object serialization (*i.e.*, implementing `Serializable`). This will cause very large messages to be sent and received, with unnecessarily large message overhead.
- Please do not assume that a fixed number of DHT operations will be invoked on your system. Your implementation should not hardcode the number or types of operations in any way.

- Remember that there is a cap on the number of AsyncTasks that you can execute simultaneously. Plan accordingly, to keep the total number of threads that you require under this limit (which is about five).
- If the testing program cannot install your APK on one or more emulators, try each of the following steps:

    1. Restart the problematic emulator.
    2. Clean your build in Android Studio, then invoke the "Build APK" menu item and try again.
    3. Manually install your APK using `adb install` to ensure that it is installable.