CSE 486/586 Distributed Systems Project 2

Simple Key-Value Storage

Update 2

There are some clarification and dropping of a requirement.

1. Your content provider and your activity should not communicate in any way except through the insert/query interface. Again, there should be no out-of-band communication between your content provider and your activity; your activity should strictly use the content provider interface to communicate with it. Otherwise, it will affect your score significantly.
2. We are dropping the requirement of returning results directly to the original requester. Here's the new description on that part from Step 1, #5---*"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."*

Update 1

There are a few minor updates.

1. New URI: in order to have a different URI from project 1's, the URI has been changed: "content://edu.buffalo.cse.cse486_586.simpledht.provider"
2. Eclipse project name: in order to simplify grading, we have decided to fix the Eclipse project name. Please use SimpleDHT (which will generate SimpleDHT.apk) as your Eclipse project name.
3. Package name: in order to simplify grading, we have decided to fix the package name as well. It should be "edu.buffalo.cse.cse486_586.simpledht".
4. Join requests: the first emulator instance, emulator-5554, should receive all join requests first. Your implementation should not choose a random node to do that.
5. It is mandatory that each group shares one design document, though each member of your group should submit one copy individually (for fault tolerance). For grading, we will randomly pick one copy of your group's design document and grade it; everyone in your group will receive the same score for the design document.

Introduction

In this assignment, you will design a simple distributed key-value storage 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: http://www.cse.buffalo.edu/~stevko/courses/cse486/spring12/lectures/dht.pptx
2. Chord paper: http://conferences.sigcomm.org/sigcomm/2001/p12-stoica.pdf

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.

Package Name

For easier grading, we will fix the package name. Your package name should be “edu.buffalo.cse.cse486_586.simpledht“.

Hash Function

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. 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 0: Writing the Main Activity

Your app should have an activity used for *testing.* It should have two buttons, one button that displays "Test" and the other button that displays "Dump." Here are the requirements for the buttons.

1. Test
    a. When touched, this button should insert ten <key, value> pairs into your DHT and retrieve them by using your content provider's insert() and query().
    b. Each of the ten insert operations should be preceded by a 1-second delay. This is to prevent potentially concurrent inserts.
    c. After generating all insert operations, all <key, value> pairs just inserted should be retrieved. Again, each of the ten queries should be preceded by a 1-second delay.
    d. Each retrieval should display the <key, value> pair retrieved on the screen. The format of the display is "<key, value>" as a string.
    e. The format of the <key, value> pairs is the following:
        i. Key: the sequence number represented as a string starting from 0 (e.g., "0", "1", "2", etc.)
        ii. Value: string "Test" concatenated by each sequence number (e.g., "Test0", "Test1", "Test2", etc.)
    f. Each touch of the button resets the sequence number to 0.
    g. Thus, the sequence of operations should be the following:
        i. 1-second delay
        ii. Insert <"0", "Test0">
        iii. Increment the sequence number and repeat the above insertion with the 1-second delay until <"9", "Test9">.
        iv. 1-second delay
        v. Query <"0", "Test0">
        vi. Display "<0, Test0>"
        vii. Increment the sequence number and repeat the query & display with the 1-second delay until <"9", "Test9">.
2. Dump
    a. When touched, this button should dump all the <key, value> pairs *stored in your local storage*. Since you need to implement a distributed key-value storage based on Chord, each instance stores <key, value> pairs that belong to one partition in the Chord ring. This button should display all local <key, value> pairs on the screen.
    b. The order of <key, value> pairs you display does not matter as long as it shows all locally stored <key, value> pairs.

Step 1: Writing the Content Provider

Along with the main activity, 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. ***At the minimum, your content provider should implement insert() and query().*** The interface definition is the same as the previous assignment (see below), which allows a client app to insert arbitrary <"key", "value"> pairs where both the key and the value are strings. *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. ***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.
6. ***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.* 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.
7. ***To avoid confusion, we will again fix all the port numbers.***
    a. Your app should open one server socket that listens on 10000.
    b. You need to use run_emu.sh and set_redir.sh to set up the testing environment. Because of this, if you run 5 instances, the redirection ports are 11108, 11112, 11116, 11120, & 11124.
    c. You should just hard-code the above 5 ports and use them for communication in your final submission.
8. ***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.
9. ***Your content provider's URI should be*** "content://edu.buffalo.cse.cse486_586.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 (though for your "dump" button, you *might* want to match something like "content://edu.buffalo.cse.cse486_586.simpledht.provider/*", but this is not required and up to you).
10. ***Your content provider should have two columns.***
    a. The first column should be named as "provider_key" (an all lowercase string). This column is used to store all keys.
    b. The second column should be named as "provider_value" (an all lowercase string). This column is used to store all values.

c. All keys and values that your provider stores should use the string data type.

11. Since the column names are “provider_key” and “provider_value”, any app should be able to insert a <key, value> pair as in the following example:

```
ContentValues keyValueToInsert = new ContentValues();

// inserting <”key-to-insert”, “value-to-insert”>
keyValueToInsert.put(“provider_key”, “key-to-insert”);
keyValueToInsert.put(“provider_value”, “value-to-insert”);

Uri newUri = getContentResolver().insert(
   providerUri,    // assume we already created a Uri object with our provider URI
   keyValueToInsert
);
```

12. Similarly, any app should be able to read a <key, value> pair from your provider with query(). Since your provider is a simple <key, value> table, we are not going to follow the Android convention; your provider should be able to answer queries as in the following example:

```
Cursor resultCursor = getContentResolver().query(
   providerUri,    // assume we already created a Uri object with our provider URI
   null,           // no need to support the projection parameter
   “key-to-read”,    // we provide the key directly as the selection parameter
   null,           // no need to support the selectionArgs parameter
   null            // no need to support the sortOrder parameter
);
```

Thus, your query() implementation should read the *selection* parameter and use it as the key to retrieve from your table. In turn, the Cursor returned by your query() implementation should include only one row with two columns using your providers column names, i.e., “provider_key” and “provider_value”. You probably want to use android.database.MatrixCursor instead of implementing your own Cursor.

13. ***Your content provider should store the <key, value> pairs that belong to its own partition using one of the data storage options.*** The details of possible data storage options are in http://developer.android.com/guide/topics/data/data-storage.html. You can choose any option; however, the easiest way to do this is probably use the internal storage with the key as the file name and the value stored in the file. *Note that your content provider should only store the <key, value> pairs local to its own partition.*

Design Document

Your group needs to write a design document of up to 2 pages (12 pt font in .pdf). This should include:

1. What components your group designed and what they do
2. The description of what procedure your group follows in order to update predecessor and successor pointers. Please include an English description as well as pseudo-code.
3. Any special things to note.
4. *The whole name of your package to facilitate installing/uninstalling of your apk for grading*

Your group should share one design document, though everyone should submit a copy of your design document for redundancy.

Project Group

Your group should share one design document, though everyone should submit a copy of your design document. We will just pick one copy and grade it. However, coding should be done individually; please write your own code.

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 the following three files:

- Your app’s .apk
- Your group’s design document in .pdf. Please note that everyone should submit the design document individually even though your group writes the document together. This is to make sure that we do not miss anything from you.
- Your entire Eclipse project source code tree zipped up in .zip. To do this, please go to your Eclipse workspace directory, find your app, and zip it.

Deadline: **4/13/12 (Friday) 2:59pm**

This is right before class @ 3pm. The deadline is firm; if your timestamp is 3pm, it is a late submission.

Grading

This assignment is 15% of your final grade. The breakdown for this assignment is:

- 10%: your app
  - 1% if the dump button works correctly (apart from the underlying insert/query over the DHT)
  - 1% if the test button works correctly (apart from the underlying insert/query over the DHT).
  - 5% if insert and query operations work correctly (with a static/stable membership)
  - 3% if node join works correctly
- 5%: your design document
  - 3% for the correct and clear description of your predecessor/successor update procedure
  - 2% for the overall clarity of the rest of the description

Late Submission Policy

- Late submissions will result in a 20% penalty per day, e.g., (your regular score * 80 / 100) for the first day, (your regular score * 60 / 100) for the second day, etc. A day is defined as 24 hours after the day/time the assignment is due (excluding weekends or school holidays).
- No help will be available from the TAs or from the instructor for a project after its scheduled due

date.
● After five (5) days, the assignment will no longer be accepted.

Academic Integrity

● Under any circumstance, you should not copy others' code.
● You need to get permission first when you copy from other sources, e.g., the Web, books, etc. If you get the permission, then you also need to clearly comment in your code which part you copied. This is for your protection; otherwise, the course staff might not know whether you have gotten the permission or not.
● The exception is Android Developers. It contains many useful coding examples, and you are free to copy any code from there. However, **you still need to clearly comment in your code that you copied from the developer website.** Again, this is for your protection.