CSE 486/586 Distributed Systems Project 3

Replicated Key-Value Storage

Update 1

We have slightly reworded one of the requirements for the design document in order to help you understand what you need to consider even from your design stage. It is #3 and reworded as the following:

> “
>
> *3. A discussion of what consistency model your group is aiming to achieve as a result of the failure cases you consider:* The way you handle failures and how your system behaves under failures greatly affects what consistency you can provide. Please carefully write this part and include what consistency your group is aiming to achieve by considering the failure cases you discuss in #2 above.
>
> ”

Introduction

At this point, most of you are probably ready to understand and implement a Dynamo-style key-value storage; this assignment is about implementing a simplified version of Dynamo. (And you might argue that it’s not Dynamo any more ;-) There are three main pieces you need to implement: 1) Partitioning, 2) Replication, and 3) Failure handling.

This document assumes that you are already familiar with Dynamo. If you are not, that is your first step. There are many similarities between this assignment and the previous assignment for the most basic functionalities, and you are free to reuse your code from the previous assignment.

References

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

1. Lecture slides on Dynamo: http://www.cse.buffalo.edu/~stevko/courses/cse486/spring12/lectures/dynamo.pptx
2. Dynamo paper: http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

The lecture slides give an overview, but do not discuss Dynamo in detail, so it should be a good reference to get an overall idea. The paper presents the detail, 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 Dynamo. Mainly, the following are the things you ***do not*** need to consider from the Dynamo paper.

1. Virtual nodes (mainly Section 4.2, but any discussion about virtual nodes in the paper in general): we do not use virtual nodes when creating partitions. Just like the previous assignment, all nodes are physical.
2. Partitioned network (any discussion about it in the paper): we do not deal with network partitions in this assignment (although we do consider individual failures).
3. Data versioning (Section 4.4): we only keep the most recent object.
4. Hinted handoff (Section 4.6): we do not consider hinted handoff.
5. Adding/removing nodes (Section 4.9): we assume that the global membership is static; individual failures may occur, but only temporarily.

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

Project & Package Names

1. Eclipse project name: in order to simplify grading, please use SimpleDynamo (which will generate SimpleDynamo.apk) as your Eclipse project name.
2. Package name: again for grading, please use “edu.buffalo.cse.cse486_586.simpledynamo“ as your package name.

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 five buttons, three buttons that displays “Put1”, “Put2”, and “Put3”, one button that displays “Get”, and the last button that displays “Dump.” Here are the requirements for the buttons.

1. Put* buttons

a. All Put* buttons should operate the same way except that they insert different values with the same set of keys.
b. When touched, it should insert ten <key, value> pairs into your storage by using your content provider's insert().
c. Each operation should be preceded by a 3-second delay.
d. Each touch of the button resets the sequence number to 0.
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 for each button concatenated by each sequence number.
1. For the button "Put1", the values should be "Put10", "Put11", "Put12", etc.
2. For the button "Put2", the values should be "Put20", "Put21", "Put22", etc.
3. For the button "Put3", the values should be "Put30", "Put31", "Put32", etc.

2. Get
a. This button should retrieve ten keys, "0", "1", …, "9" with their corresponding values using your content provider's query() interface. Again, each query should be preceded by a 3-second delay.
b. Each retrieval should display the <key, value> pair retrieved on the screen. The format of the display is "<key, value>" as a string.

3. 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 Dynamo, each instance stores <key, value> pairs that belong to one partition as well as replicas from other partitions. 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.
c. Note that each local storage in this assignment should contain replicas as well. Obviously, the Dump button should also display these replicas when touched.

Step 1: Writing the Content Provider

Along with the main activity, your app should have a content provider. Just like the previous assignment, this content provider should implement all storage 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. The following are the requirements for your content provider:

1. ***Membership***
a. *Just as the original Dynamo, every node should know every other node.* This means that each node knows all other nodes in the system and also knows exactly which partition belongs to which node; any node can forward a request to the correct node without using a ring-based routing.
b. Unlike the original Dynamo, you can assume that there are *always 5 nodes in the system*. There is no need to implement adding/removing nodes from the system.
c. However, there can be *at most 1 node failure at any given time*. We will discuss this in detail later.
2. ***Request routing***
a. Unlike Chord, each Dynamo node knows all other nodes in the system and also knows exactly which partition belongs to which node.
b. Under no failures, all requests should be directly forwarded to the coordinator, and the coordinator should be in charge of serving read/write operations.
3. ***Quorum replication***
a. *The replication degree N should be 3.* This means that given a key, the key's coordinator

as well as the 2 successor nodes in the Dynamo ring should store the key.

b. *Both the reader quorum size R and the writer quorum size W should be 2.*

c. The coordinator for a get/put request should *always contact other two nodes* and get the votes. For each request, the coordinator should return to the requester whether the request was successful or not. If the coordinator fails to obtain the minimum number of votes, it should return an error.

4. ***Failure handling***

a. ***Warning: this part is probably the main difficulty of this assignment.*** Handling failures should be done very carefully because there can be many corner cases to consider and cover. Before you start writing your code, first discuss among your group and make sure that you know what you want to do.

b. *Please focus on correctness rather than performance.* Once you handle failures correctly, if you still have time, you can improve your performance.

c. *At any point of time, there can be at most one node failure.* We will emulate a failure only by force closing an app instance. We will ***not*** emulate a failure by killing an entire emulator instance. You can force close an app by going to “Settings” -> “Applications”, then select the app. There’s a “force close” button.

d. *There is no need to implement a failure detector;* just as the original Dynamo, each request should be used to detect a node failure.

e. *For this purpose, you can use a timeout for a socket read;* you can pick a reasonable timeout value, e.g., 100 ms, and if a node does not respond within the timeout, you can consider it a failure.

f. ***Do not rely on socket creation or connect status to determine if a node has failed.*** Due to the Android emulator networking setup, it is not safe to rely on socket creation or connect status to judge node failures. Please use an explicit method to test whether an app instance is running or not, e.g., using a socket read timeout as described above.

g. Just as the original Dynamo, when a coordinator for a request fails and it does not respond to the request, *its successor should be contacted next for the request.*

h. There is no need to implement hinted handoff; during a write request, if there is a failed node, *your implementation should simply skip the failed node for the write operation.*

i. *All failures are temporary;* you need to assume that a failed node will recover soon. *When a node recovers, it should copy all the object writes it missed during the failure.* This can be done by asking the right nodes and copy from them.

5. ***Your content provider’s URI should be*** “content://edu.buffalo.cse.cse486_586.simpledynamo.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.simpledynamo.provider/*”, but this is not required and up to you).

6. ***Same requirements as the previous assignment***

a. ***The content provider should implement all functionalities.***

b. ***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.*

c. ***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 storage in order to find the correct position in the Dynamo ring.*

d. ***To avoid confusion, we will again fix all the port numbers.***

i. Your app should open one server socket that listens on 10000.

ii. 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.

iii. You should just hard-code the above 5 ports and use them for communication in your final submission.

e. ***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.

f. ***Your content provider should have two columns.***

i. The first column should be named as “provider_key” (an all lowercase string). This column is used to store all keys.

ii. The second column should be named as “provider_value” (an all lowercase string). This column is used to store all values.

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

g. 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
);
```

h. 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.

i. ***Your content provider should store the <key, value> pairs that belong to its own partition (including replicated <key, value> pairs) 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 that belong to its own partition including replicated <key, value> pairs.*

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 your group's strategy is to handle node failures:* Please carefully write this part and include what cases you consider and how you address those cases.
3. *A discussion of what consistency model your group is aiming to achieve as a result of the failure cases you consider:* The way you handle failures and how your system behaves under failures greatly affects what consistency you can provide. Please carefully write this part and include what consistency your group is aiming to achieve by considering the failure cases you discuss in #2 above.
4. Any special things to note.
5. *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: **4/30/12 (Monday) 11:59pm**

Note that this is not our usual deadline time of 2:59pm. The deadline is firm; if your timestamp is 12am, 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 put, get, and dump buttons work correctly (apart from the underlying storage implementation)
    - 3% if insert/query/replication works correctly under no failures
    - 3% if insert/query/replication works correctly under one failure
    - 3% if node recovery works correctly
- 5%: your design document
    - 2% for the correct and clear description of your failure handling
    - 2% for the correct and clear description of the consistency model your group tries to achieve and why
    - 1% 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.