CSE 486/586: Distributed Systems

Programming Assignment 4[1]
Replicated, Distributed Key-Value Storage

Introduction

For this assignment you will design and implement a simplified version of the Amazon Dynamo key-value storage system. (The simplifications are such that one would probably not consider it Dynamo any more!) You will have to implement partitioning, replication, and failure handling.

Your goal is to implement a distributed key-value storage system that provides both availability and linearizability. Your implementation must be able to perform successful read and write operations even in the presence of failures.

This document provides an implementation guideline for achieving a successful project. *However, you are free to come up with your own design in many parts of the project, provided that you provide both availability and linearizability at the same time in a way that the tester can understand.* You must, however, implement partitioning and replication in exactly the manner that Dynamo does (so that the tester can understand your implementation).

This document assumes that you are already familiar with Dyanmo. If you are not, you should first go and read the Dynamo paper (which is required reading for the course) and review the Dynamo slides from class.

This assignment has many similarities to the previous assignment, your Chord ring implementation, and you may draw from your Chord implementation (or other previous projects) as appropriate. Please remember to cite any code that is taken from external references, such as Android Developers or the Oracle Java tutorials.

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 data replication
- Implement data partitioning
- Handle node failures while continuing to provide availability and linearizability

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

[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.

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 `SimpleDynamo`. You can find YouTube tutorials detailing the process here.

2 The Content Provider

The graded portion of this project is a `ContentProvider` providing simplified Dynamo functionality to store and retrieve data across multiple devices even under failures. The provided template uses the package name `edu.buffalo.cse.cse486586.simpledynamo`, 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.

Like the SimpleDht project, all functionality for the requirements for this project should be provided by your content provider. It should handle availability in the face of failures via replication, as well as linearizability, independent of any Activity you may implement. The Activity in this project is used for your testing purposes only.

You will use SHA-1 as the hash function to generate IDs for the system, the same as with our Chord implementation. You should use the `genHash()` function from Project 3 for this purpose. 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

In implementing your content provider, you may make the following assumptions:

1. There will always be 5 nodes in the system (except for failures). There is no need to implement joining or departure of nodes. Unlike the previous project, your nodes may hard-code the ring structure, as long as they handle failure.

2. There will be *at most one failure at a time*. We will emulate a failure by force-closing an app on one emulator. We will not close the emulator itself.

3. All failures are temporary. You can assume that a failed node will recover soon. It will not be permanently unavailable during a run.

4. Correctness is more important than performance! *Once you handle failures correctly,* if you have time, you can improve performance.

5. You do not need to implement virtual nodes. (These are part of the Dynamo protocol, see the Dynamo paper!) All partitions are static and fixed.

6. You do not need to handle hinted handoff. This means that, during a failure, it is OK to replicate on only two nodes (if the third replica would be on a failed node). (Again, see the Dynamo paper.)

7. Linearizability will be checked only on a *per-key basis.* You do not need to provide any consistency guarantees between keys. Formally, you must implement *per-key linearizability*.

2.1.1 Requirements

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

2. The content provider should implement the Dynamo-like functionality described here. Like the previous assignment, this includes all communication and functionality for the replicated store, including creation of appropriate tasks and sockets, response to requests from other AVDs, *etc*. No functionality for the Dynamo store should be provided by your Activity, which is used only for your own testing.

3. When a node recovers from failure, it should copy all of the object writes that it missed during the failure. This can be done by making requests to the appropriate nodes and copying their data.

4. Your content provider must support *concurrent read and write operations.*

5. Your content provider must handle a failure occurring *during* read and write operations.

6. Replication in your content provider should be handled just as it is in Dynamo. Each key-value pair should be replicated over three consecutive partitions, starting from the partition that the key belongs to (based on its consistent hashing).

7. All replicas must *store the same value for each key*. This is *per-key linearizability*.

8. 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 Dynamo ring.

9. Any app using your content provider can submit arbitrary key-value pairs of Java strings (*e.g.*, <"I want to", "store this">), query arbitrary keys and retrieve their values, or delete arbitrary keys.

10. As in the SimpleDht assignment, your app must support query operations for the special keys "@" and "*", which are defined as before.

11. 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.

12. 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 ID space, and key-value pairs it is required to replicate from other partitions!*

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.* You can (and should) use it to evaluate your Dynamo implementation as you see fit.

2.3 Implementation Guidelines

The following are some guidelines for implementing your content provider. Remember that you are free to implement your project however you wish, as long is it meets the project requirements! These guidelines are provided only to assist you in designing a successful implementation. In particular, many parts of these guidelines can be improved upon using techniques from the lectures or literature, if time allows. Remember that your partitioning and replication behaviors must be exactly as specified above, whatever techniques you choose.

Membership. Just as in the original Dynamo paper, every node can know about every other node, including its location and partition of the ring. This means that any node can forward a request that it cannot handle locally directly to a node that can without using ring-based routing.

Request routing. When there are no failures, a request can be forwarded directly to the coordinator for the key (*i.e.*, the key's successor), and the coordinator should handle read/write operations for that key.

Quorum replication. In order to implement linearizability, you can implement quorum-based replication. Note that the original design does *not* handle linearizability, and you will need to adapt it. If you implement quorum replication, the replication degree (N) should be 3, and the reader and writer quorum sizes (R and W) should be 2. This means that, given a key and no failures, the key's coordinator as well as the next two successor nodes in the ring should store the key and its value, and that the coordinator should, for every request for a get or put, contact *two other nodes* and get a vote from each. You can *version* objects in order to distinguish stale objects from the most recent copy (*e.g.*, following failure and re-joining of a node). Upon read, if more than one version of an object is returned, the coordinator should use the most recent version.

Chain replication. Another alternative for implementing linearizability is chain replication. The details for this can be found in *Chain Replication for Supporting High Throughput and Availability*, by Robbert van Renesse and Fred B. Schneider, found at `http://www.cs.cornell.edu/home/rvr/papers/OSDI04.pdf`. In chain replication, a write is always made to the first partition, and then propagates to the next two partitions in order. The last partition returns the result of the write. Read operations are always made to the last of the chain of partitions.

Failure handling. Failures must be handled very carefully, as there may be many corner cases to consider and cover. Just as described in the Dynamo paper, all messages between nodes can be used to detect failures. You can use a timeout on socket operations (in particular, read operations), just as we did in the GroupMessenger2 project for this, using a reasonable timeout period (*e.g.*, 100 ms) and considering a node failed if it does not respond before the timeout. Just as in the previous project, *do not rely on socket creation or connection status to detect failure,* as our emulated environment may distort this signal. When the coordinator for a request fails and it therefore does not respond to a request, its successor can be contacted next to fulfill the request. Remember that you can assume *at most one failure at a time.*

2.4 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.

You should also test your program yourself by writing your own tests and drivers.

The testing program for this project is divided into six phases. The first three phases will not take much time compared to the second three, so you should focus on them first and finish them as quickly as possible. This will allow you to spend as much time as needed on the last three phases.

The phases are:

1. Testing basic operations

    - This phase will test insert, query, and delete (including the special keys @ and *). This will ensure that all data is correctly replicated. It will not perform any concurrent operations or test failures

2. Testing concurrent operations with differing keys

    - This phase will test your implementation under concurrent operations, but without failure.
    - The tester will use different key-value pairs inserted and queried concurrently on various nodes.

3. Testing concurrent operations with like keys

    - This phase will test your implementation under concurrent operations with the same keys, but no failure.
    - The tester will use the same set of key-value pairs inserted and queried concurrently on all nodes.

4. Testing one failure

    - This phase will test every operation under a single failure.
    - For each operation, one node will crash before the operation starts. After the operation is done, the node will recover.
    - This will be repeated for each operation.

5. Testing concurrent operations with one failure

    - This phase will execute various operations concurrently and crash one node in the middle of execution. After some time, the node will recover (also during execution).

6. Testing concurrent operations with repeated failures

    - This phase will crash one node at a time, repeatedly. That is, one node will crash and then recover during execution, and then after its recovery another node will crash and then recover, *etc*.
    - There will be a brief period of time between each crash-recover sequence.

Each testing phase is quite intensive, and will take quite some time to finish. Therefore, the tester allows you to specify which phase you want to test on the command line so that you need not wait for all tests every time. However, you will still need to ensure that you run the tester *in its entirety* before you submit. *We will not run the testing phases separately in our grading.*

- You can specify which phase of the test you wish to run with the `-p` or `--phase` argument to the tester.
- You can see the available arguments for the tester with the `-h` option.

Note that if you run an individual phase with `-p` or `--phase`, the tester will always perform a fresh install. However, when running all phases, the tester will perform a fresh install only before phases 1 and 2. This means that from phase 2 onward, *all data from previous phases will remain intact.*

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 2018-05-11 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, like the previous project, *this deadline has been pre-extended, and it will not be extended any further.* This deadline *cannot be extended* due to the end of the semester!

5 Grading

This assignment is 20% of your final grade. Credit for this assignment will be apportioned as follows:

- Phase 1: 3%
- Phase 2: 4%
- Phase 3: 3%
- Phase 4: 5%
- Phase 5: 5%
- Phase 6: 3%

Thus, an application passing all six phases would receive 20 total points, representing 20% of the 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.