14-848 - Cloud Infrastructure (Fall 2019)
Project 2: Bigtable-Like Database

Important Dates

- Project Release: 9 Oct 2019
- Checkpoint 1: 18 Oct 2019
- Final Submission: 31 Oct 2019 (Trick or Treat?)

Overview

This project asks you to implement a Bigtable-like database. The basic model is described in this paper from Googlers:

- https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

The goal is to give you first hand experience building a "Big Data" database with the goal of sharpening your design and implementation skills – but, more importantly, really helping you to understand the mechanisms and design trade-offs at work by building and interacting with them.

A Few Very Important Notes

- We've simplified the project in some minor ways to allow it to fit into about three week's time, but it is still a real project. It is intended to be done by two people over the full time allocated to the project. Maybe you can do it over two weeks time if you are focused. But, if you try to pull it off in less time than that, you really won't be happy, learn as much, or have nearly as good an experience.

- Please read the paper and pay particular attention to the October 9th lecture. This project description is intended to be read in the context of the paper and in class conversation. It probably won't make much sense in isolation.

- This project is intended to be done in pairs. Working alone is really, really not recommended. You won't learn as much because the only ideas you'll have will be your own. And, you won't be as happy because you won't have any someone to share the journey with. And, you'll experience more frustration, because you'll be all alone in the inevitable tarpit, with no help getting out.

- As a shorthand, many of the test scripts use the abbreviations "cp1" and "cp2". "cp1" is the checkpoint. "cp2" is the final deliverable.

- The starter code, test code, scripts, and other machinery are brand new this year. They will inevitably require some break-in. Please do feel very free and uninhibited to ask questions. Also, we've pinned a post on Piazza called "Bigtable Errata" where we will post any updates and clarifications. *Pretty please check that post regularly. Thanks in advance!*

Continued on next page

1. Logistics

1.1 First Steps

Please read this entire document, review the lecture, read the paper, and understand the project requirements before starting development. Design decisions you make for the tablet server in Checkpoint 1 will affect the overall system for the Final Submission. If you don't have a good high-level picture before you start, you'll almost certainly end up redoing things. That's okay to some extent – but there might be better ways to spend your time. As the old saying goes, "An ounce of prevention is worth a pound of cure." If you have any questions, please do ask us. We are here to help. Really!

1.2 Get the starter code from the course website

Get the starter code from the course website and review it. Understanding what we've given you will be really helpful to helping to get started.

1.3 Development Environment

- We will be compiling and grading on CMU's Linux timeshares , the machines that are served by "unix.andrew.cmu.edu". Your submission has to run without requiring any additional installation or configuration on these hosts.

- Should you want to use libraries, packages, etc, that is okay. But, they need to be packaged with your submission so that we don't need to take any special steps to install them for grading purposes.

- Although you may write your own code in any language, our test scripts and sample codes is written in Python3. If these scripts give you trouble, please check that you are using Python 3.x vs Python 2.x, e.g. "python –version". Depending upon your own account configuration, you may need to invoke it as "python3" vs as "python".

- The external dependencies required for the grading script are already installed on unix.andrew.cmu.edu machines. Should you choose to develop and test in another environment, you will likely need to install the requests package. As a reminder, should you choose to do this – you should still test frequently and before submission on the unix.andrew.cmu.edu machines.

- Please don't run on the Shark machines, it'll take cycles away from 213/513/613 and interfere with performance measurements that are critical to their work.

- The requirements of the project are designed to be satisfied in an environment with a general purpose distributed file system shared among hosts. The project can not be adequately tested on a single host. The unix.andrew.cmu.edu timeshares provide such an environment, e.g.

- “unix4.andrew.cmu.edu”, “unix5.andrew.cmu.edu”, “unix6.andrew.cmu.edu”, “unix7.andrew.cmu.edu” and “unix8.andrew.cmu.edu”.

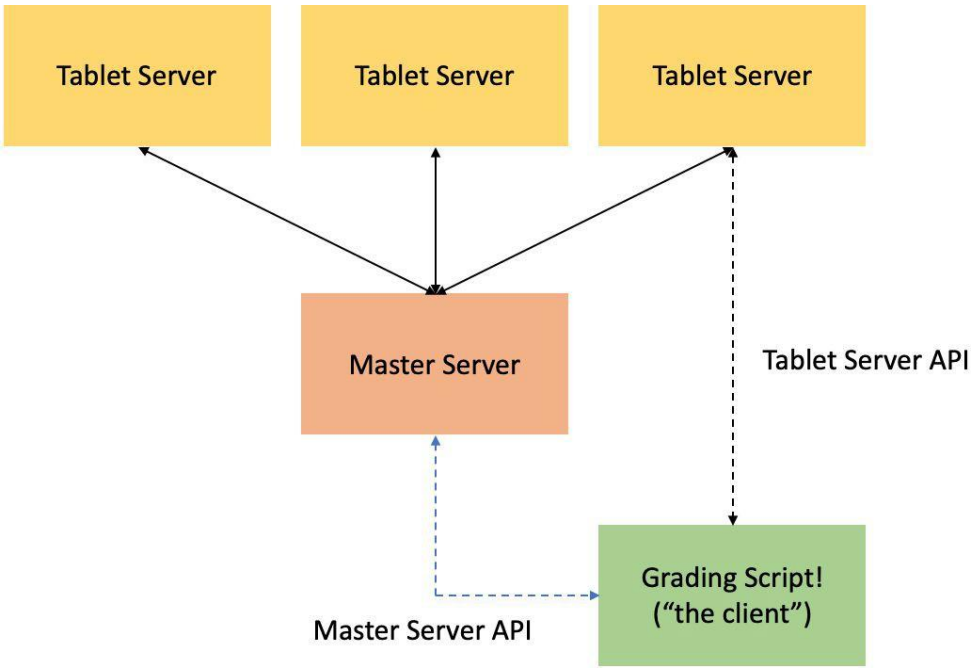
1.4 Starter Code

In the starter code, you will find:

/project_root

- dataset/
  - Sample datasets for your use
- doc/
  - These are the Markdown files that specify the REST APIs the grading scripts expect from your servers
- grading/
  - Grading scripts =)
- Makefile
- hosts.mk
- sample_server.py
  - This demonstrates how to create a webserver exposing a REST API

1.4 Requirements



REST API

We expect your master and tablet server to expose the REST APIs specified. This is how clients using your Bigtable implementation will make requests. The grading script uses these APIs. By designing the project this way, we were able to make it language independent for you.

Makefile

We will be running your tablet and master servers using the Makefile. You have to modify variables MASTER_CMD and TABLET_CMD so that the appropriate servers run correctly when we run `make master` or `make tablet`.

As shown in the `master` and `tablet` build targets in the Makefile, the master and tablet servers require certain command line arguments. Your server should accept and use these values; they cannot be hardcoded.

// THIS IS ONE LINE
// <master_hostname> <master_port> should be ignored by the tablet in Checkpoint 1
<tablet_cmd> <tablet_hostname> <tablet_port> <master_hostname> <master_port>

<master_cmd> <master_hostname> <master_port>

hosts.mk

You can modify this file to assist you when testing. When grading, we will use a different copy.

1.5 Simplifications

- Any convenient file system or shared file system may be assumed to have GFS/HDFS like properties. So, for example, if working on unix.andrew, AFS may be assumed to have HDFS like properties.
- Any convenient file system may be assumed to have atomic writes for the purpose of the Write Ahead Log (WAL) and other coordinating metadata.
- Compression and compaction are not required
- The table may be sliced just horizontally (by row), rather than horizontally and by column family as might otherwise be the case.
- You may assume that the front-end server will not fail.
- Do *not* use any other databases, e.g. SqlLite, for any parts of this project.

1.6 Constants

Sharding

- Each tablet server shards when a table/tablet contains more than a thousand (1000) row keys.

Garbage Collection

- For each cell, keep the last five (5) inserted values.

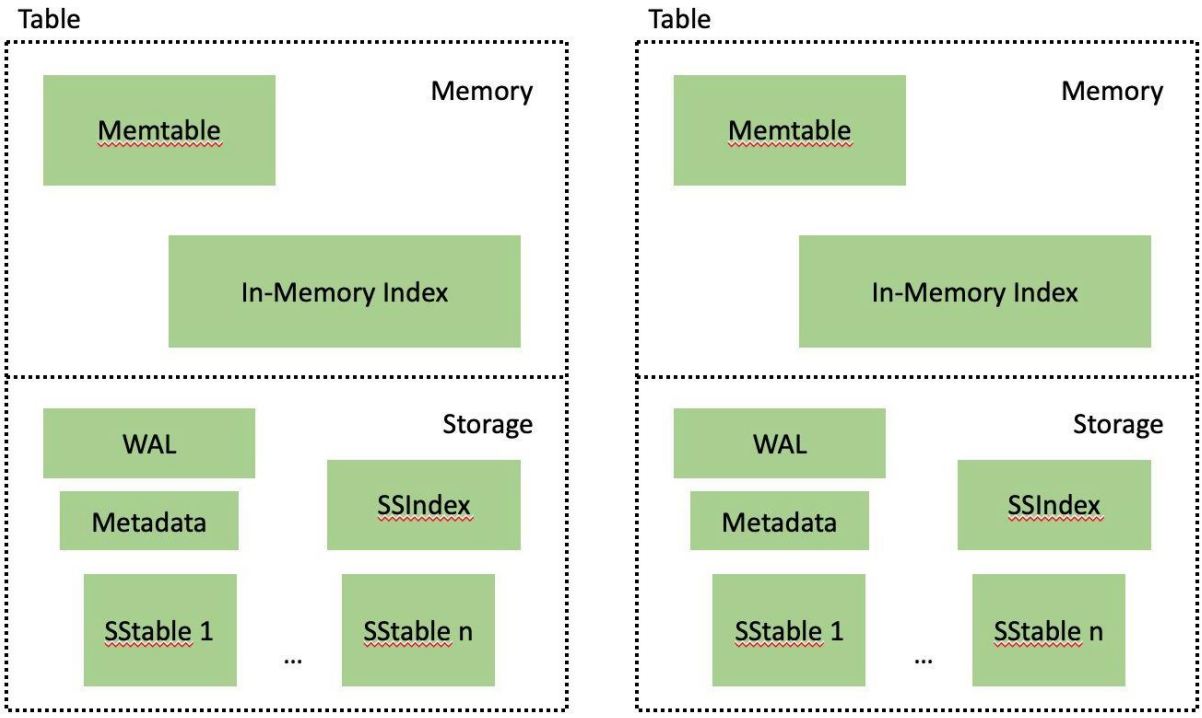
Continued on next page

Memtable

- The Memtable should be spilled to disk when there are a hundred (100) unique row keys. May we suggest making this number a configuration constant, both as a good software engineering practice and also to enable you to quickly change to a smaller valuer to facilitate testing.

2. Checkpoint 1

In Checkpoint 1, you implementing the tablet server. A tablet server may hold multiple tables, or multiple tablets (regions of tables) if sharding has taken place.



2.1 Standard Operations

Since the master server isn't available yet, creation and deletion of tables will be done by the tablet server for Checkpoint 1. A single tablet server should be able to hold multiple tables (or tablets - regions of tables).

Look at doc/tablet.md for the REST APIs that will be used for this test.

2.2 Garbage Collection

Since you've read the paper, you know that there are many potential ways to handle garbage collection in Bigtable.

For this assignment, just keep the latest **five(5)** versions of data in each cell. For example, if we insert the values (in-order): one, two, three, four, five, and six for a single cell, a retrieval of the cell should return two, three, four, five, and six.

Continued on next page

2.3 Recovery

- If the tablet server shuts down, normally or abnormally, upon restart it should reconstruct the server state using WAL and SSTables maintained in the persistent storage. A single tablet server may contain many tables. Have a WAL for each table hosted on the tablet server.

- Although the design and implementation of the SSIndexes and SSTable is up to you, please do keep in mind that you'll need to be able to recover from failure. You'll probably want to use the metadata file for recovery.

- When designing this, keep the sharding and recovery requirements of the Final Submission in mind.

- When testing, we will kill and restart your table server.

2.4 Internal Structure

- Memtables and SSTables are a fundamental part of Bigtable, and their use is required for this assignment. The number of entries in the Memtable should be configurable - there is an API call for this.

- For example, if max_entries is set at 20, and there are 19 entries in the Memtable as well as 1 SSTable already spilled and the max_entries is then set to 5, you should immediately spill 14 entries to disk, resulting in 2 SSTables on disk and 5 entries in the Memtable.

- The default memtable size is 100. Reset it to this value upon restart.

- **Important** We will also be doing code reviews of the Memtable and SSTable implementation, so just passing the grading script isn't enough to get full points for this. We'll check your design and implementation to make sure you've got a well thought out and well implemented solution, including w.r.t. this requirement.

3. Final Submission

For the final submission, you will require a master server and 3-5 tablet servers.

For this section, there are many instances where communication and coordination is required between the master and tablet servers. We do not define API calls for these - the method of communication is for you to design.

In this checkpoint, make sure you have at least two tablet servers up when running the grading scripts.

Continued on next page.

3.1 Table Creation and Deletion

In the final submission, table creation and deletion is controlled entirely by the master server. The client makes a create or delete request to the master. The master figures out which tablet servers are involved in the operation and commands the tablet servers to perform the correct operation. The master server responds to the client only when the request is complete, or completely failed). In other words, this is a *blocking* call.

There is one additional requirement for table deletion. The final submission supports locks, as described in section 3.2 just below this. Tables should be deleted only if no locks are held, otherwise the master server should return the appropriate error code.

3.2 Locking

There are many, many ways to lock. The tablet servers do not need to be aware of the locks. Clients may insert or retrieve from a table without acquiring a lock, if they don't mind the possibility of the table suddenly being deleted.

Like table creation and deletion, locking is handled entirely by the master server. The open() call gives the client a lock, the close() call relinquishes it. The client ID provided in these calls are unauthenticated and entirely arbitrary - you can assume that the grading scripts won't get up to any funny business with this.

3.3 Sharding

Sharding is the splitting of a single table amongst multiple tablet servers. There are many reasons to shard - it could be to improve throughput to a single table, or because the table is too large to be contained on a single server. *How* a table is sharded depends on the structure of the table and the expected use case.

For the assignment, we once again arbitrarily decide to shard a table or tablet whenever there are at least **1000 rows** contained on the server. As usual, please make this a configuration constant not a "magic number". Exactly how this is done is up to you, as long as the follow requirements are met:

1. Sharding should not interrupt the operations of a client actively using the server. For example, a client performing many insert operations should not encounter rejection when the tablet server reaches shard limits - the tablet server should shard in the background and forward any requests as needed.
2. The master server should be kept updated. When a shard operation is complete, the master server should understand which tablet server holds which region of the table.
3. Sharding should split the row set in a reasonably equal manner. For example, if the pre-shard table contains 1000 unique keys [1, 1000], the post shard tables should contain keys of the approximate range [1, 500] and [501, 1000].

These requirements do allow a client to ignore sharding and continue performing inserts to a single tablet server. For this assignment, assume that clients are well behaved and will check in occasionally with the master server when performing excessive numbers of inserts, so that system performance is not degraded.

Continued on next page.

3.4 Recovery

Bigtable should be able to handle the permanent loss of a tablet server - that's why we have the WALs and SStable spills stored on a distributed file system. The master server should frequently check in on the tablet servers. In the event of an extended loss, the master server should have other tablet servers take over the tables/regions held by the lost tablet, using the WALs and SSTables left in shared storage.

To test this, we will kill a single tablet server. For the data sets provided, we expect recovery to be complete within **one minute**.

4. Grading

4.1 Checkpoint 1

Overall Operations	20
Recovery	10
Memtable/SSTable	10
Total	**40**

As noted above, we will review the implementation of Memtable and SSTable. Passing the grading script is not sufficient for that section.

4.2 Final Submission

Master Server	15
Sharding	20
Recovery	20
Style	5
Total	**60**

We don't have a specific style guide, given that we don't specify a language. General rules of thumb:

- Obey the style guidelines common for the language you are using
- If the course staff loses their minds reading your code or trying to run your servers, you are likely to lose style points.

5. Submission

Zip your folder from /project_root and submit the zipfile to Canvas. Make sure that upon unzip, we get a project folder with all required files within. Please include a file called "PARTNERS" with the full names and @andrew.cmu.edu email addresses for both partners.