Project 4: GT Store

Authors: Eva Grace Bennett, Gopesh Singal

System Components

Centralized Manager

The centralized manager for the GT Store system serves as a server that interacts with the client API and monitoring the storage nodes. The centralized manager implements a consistent hashing system with a hash ring, adding the nodes to this ring when initialized in order to best apply the key-values to the nodes when hashed. When a client interacts with the manager to input a key-value pair, the manager redirects the client to its assigned node, where the node would then hold the key-value pair. Ideally, the manager would then signal the next K nodes in the hash ring to also contain the key-value pair in order to account for the case when the target node would fail, therefore preventing permanent loss of information.

Storage Node

The storage nodes in the GT Store system are responsible for holding the key-value pairs for the clients interacting with the system. The storage node is capable of putting and getting the values from its own localized information, managing the client key-value pairs once they have received the information.

Client API

The client API consists of the following functions: `init`, `put`, `get`, `finalize`.

- `init`: This function initializes the client class object, focusing on initializing the channel connection to the managerial server. In doing so, the client is then capable of fully interacting with GT Store.
- `put`: This function causes the centralized manager to route the client to a storage node that has a hashing value that corresponds to the key of the key-value pair the client wished to add to GT Store.
- `get`: This function interfaces with the storage node assigned prior in order to receive the client's specific value when considering a key.
- `finalize`: As established in the documentation of the project, the finalize method did not necessitate formal implementation. For now, the method simply serves as a placeholder.

Design Principles

Data Partitioning

Through the use of the consistent hashing implementation and the hash ring. As established in the Piazza, we were allowed to use pre-existing libraries for consistent hashing, so we imported the hashing available at this GitHub repo. Through the use of the consistent hashing, the partitioning of the data would be more or less even.

Data Replication

While not formally implemented, the system for data replication would be the assignment of key-value pairs to the next K nodes within the hash ring when the manager initially receives the key-value pair from the client. In doing so, even if a storage node server were to fail, it would still be possible to recover and access the client's information.

Data Consistency

Because we assume an overwrite when the `put` method is called from the client API, the reassignment of the key-value pair would repeat the data replication step, therefore ensuring the same nodes always have the same information. Even if a node were to fail, because the nodes were assigned sequentially, this prevents the method from ever causing "gaps" where a node may have outdated information.

Client Driver Application

```
(base) glbennett8876@lawn-128-61-79-103 gtstore % ./run.sh
Begining Test 1: Basic Single Server GET/PUT
------------------------------------------------
OK, 0.0.0.0:3
Manager listening on 0.0.0.0:50051
key1, value1, 0.0.0.0:3
OK, 0.0.0.0:3
OK, 0.0.0.0:3
OK, 0.0.0.0:3
key1, value2, 0.0.0.0:3
key2, value3, 0.0.0.0:3
key3, value4, 0.0.0.0:3
./run.sh: line 23: 60168 Terminated: 15          ./start_service.sh --nodes 1 --rep 1
End Test 1
------------------------------------------------
Begin Test 2: Basic Multi-Server GET/PUT
------------------------------------------------
OK, 0.0.0.0:3
Manager listening on 0.0.0.0:50051
key1, value1, 0.0.0.0:3
OK, 0.0.0.0:3
OK, 0.0.0.0:4
OK, 0.0.0.0:1
key1, value2, 0.0.0.0:3
hello, value3, 0.0.0.0:4
world, value4, 0.0.0.0:1
./run.sh: line 51: 60188 Terminated: 15          ./start_service.sh --nodes 5 --rep 3
End Test 2
------------------------------------------------
(base) glbennett8876@lawn-128-61-79-103 gtstore %
```

`start_service.sh`: This executable file starts up the data manager and specified nodes
`./start_service.sh --nodes <num_nodes> --rep <num_replicas>`

`client.sh`: This executable file starts an instant of client. It performs either a get or put call.
`./client.sh --put <key> --val <value>`
`./client.sh --get<key>`

`run.sh`: This executable runs the test case. It starts up the service and then makes several client calls.
`./run.sh`

Design Trade-offs

We stuck to a mostly object-oriented approach where the manager and storage nodes were well-represented, each having its own server. While this makes for more understandable design, better implementation could have been possible if the servers were not one-to-one with these objects and instead incorporated usage of the objects instead. In the end, it comes down to a preference between implementation ease and understandability.

Implementation Issues

Due to the time constraint and overarching issues, the complete implementation of the system was not able to be completed, as well as the performance tests.