Project 4: GT Store

Due Date - Sunday, Apr 23, 2023, 11:59 PM

0. Reminder

- You can choose to work in pair on this project.
- Test program on VM cluster before making compression and submission.

1. Introduction

In this project, you will implement GT Store, the distributed key-value store system, by network calls for simulating the shopping processes. GT Store is a centralized manager to multiple data nodes. For the clients, the driver application, will perform gets and sets to key/value pairs while we manually take storage nodes offline to ensure that the service can recover from node failure.

System Components

1. **GT Store**
   1. **Centralized manager**: One user-space process will handle all the control path decisions of the key-value store. The manager's tasks include, but are not limited to:
      1. Membership management
      2. Load balancing of clients across replica nodes
      3. Index management. E.g., you could (but not required) allow clients first contact centralized manager on initialization to figure out the data nodes where they should send their requests.
   2. **Storage/Data nodes**: N user-space processes will act as the storage nodes in the key-value store. They will store in-memory and provide the key-value pairs, those they are responsible for. These processes represent the actual key-value store.
2. **Driver application**: A single process that interacts with GT Store. In a typical interaction, the application
   1. reads the latest version of the object
   2. mutates it, and
   3. stores it back.
3. **Client library**: The driver applications should use a simple programming interface, in order to interact with the key-value store. The base API includes;
   1. `init()`: initialize the client session with the manager and other control pathoperations.
   2. `put(key, value)`
   3. `get(key)`
   4. `finalize()`: control path cleanup operation.

For each request, the maximal key size is `20 bytes`, and the maximal value size is `1 KB`.

2. Project Details

Design Principles

1. **Data Partitioning** GTStore aims to be a highly scalable key value store, thus storing all the data in a single node will not work. Ideally, the system should be able to serve an increasing number of client requests in proportion to hardware resources we provision (number of data nodes). Data partitioning schemes divide the responsibility of managing data elements across the data nodes. For this section, you will design and implement:
    1. A data partitioning scheme to partition incoming data across data nodes.
    2. Data insert/lookup algorithm/protocol for the selected partitioning scheme in the context of your system design.
    3. Discuss pros and cons of your design.
2. **Data Replication** GTStore maintains K number of data replicas for any stored object. Replication increases fault tolerance of the system and under certain consistency schemes increase the system availability. In this section you will design and implement:
    1. A data replication mechanism for GT Store.
    2. Clearly describe the replication protocol/set of events during data insert/lookup.
    3. How does your replication mechanism works alongside with the proposed data partitioning technique? e.g. node with the primary replica fails.
3. **Data Consistency** Data consistency scheme plays a major role in defining the availability of any data store. Immediate replication is ok, and the recommended way to implement the behavior expected in the demo tests. For instance, your consistency protocol may act as follows:
    1. Inserted objects get stored on K (replication parameter) nodes within a bounded time.
    2. Read operations are guaranteed to receive the latest version of the requested object.

3. Implementation

We've provided a very basic code skeleton and run script so as to give you a usage example. Feel free to make any necessary modifications to the classes and function parameters defined inside the given `{.hpp, .cpp}`. You can also modify the Makefile. You can add files, just make sure to update the build scripts accordingly. * gtstore.hpp * manager.cpp * storage.cpp * client.cpp * test_app.cpp * Makefile * run.sh * RPC related meta-files if any

Implementation details

- Code implementation in C/C++.
- You will model each node in the distributed system using a separate process running on a single machine.
- Avoid using shared memory or file system to communicate between nodes. It has to be a network call, so your GTStore can work when deployed across many machines.
- You may use sockets for communication between nodes.
- You are also allowed to use any support for Remote Procedure Calls (RPC). Look at the Google gRPC for choice of framework. Note that if you go this route, you will likely benefit only from the stubs for the message buffer management (i.e.,sending/receiving the key value pairs), but you will still need to orchestrate the distributed interactions among the client(s) and server(s) by yourself.
- Demonstrate interaction with service
- You are not allowed to use any other third party libs without explicit permission.

4. Report & Deliverables

Your report is as important as your code implementation and should include an elaborate description of the design choices and functionality. Please include screenshots of the relevant log messages for every functionality. Additionally, address the questions described in Section 2 and Section 3 formatted in the following way:

- **System Components**
  - GTStore Centralized Manager.
  - GTStore Storage Node.
  - Client API calls (one-by-one).
- **Design Principles**
  - Data Partitioning
  - Data Replication
  - Data Consistency
- **Client Driver Application** Please create driver programs as described in the demo section
- **Design Tradeoffs** We often have to tradeoff one aspect of the system, in favor of another. Describe one interesting property of your designed data store, e.g., lightning fast responses, linearscalability, etc, and identify how that design decision impacts the performance of the rest of the system, or particular aspect of the system, such as lowered throughput, increase instorage space/replicas, etc. .
- **Implementation Issues** Describe any restrictions or faults of your current implementation, if any.

- **Performance Tests and Graphs:** Run these performance and load balance tests, and create the graphs as described.