Simple Object Storage - Report 4

Nijad Huseynov

June 2023

Introduction

This week, I mainly did a development to finalize the project. The tasks that have been successfully accomplished are listed in detail below.

Completed tasks

Primary node is implemented. Now the system can be started in a distributed fashion. It is responsible to manages volumes on data nodes.

Additionally, command line interface is designed and implemented to run the system. To implement the command line interface we have used Cobra library which makes the cmd implementation quite easy. To start the primary node, following command can be used.

```
./sos primary --port=8080 --grpc_port=1234
```

To start the data node, following command can be used.

```
./sos data --vol_dir="tmp/node1" --primary_node="localhost:1212" --port="8081"
    --node_id="1"
```

Data node is completed. Read and write apis are implemented to write and read the object.

Locks are added on volumes to prevent the volume corruption. To do that, i have used sync.Mutex primitive from the golang library. Now before each write to the volume, each gorotine should aquire lock on the volume to prevent data corruptions.

Heartbeat mechanism is implemented. The hearbeat is very important in distributed systems. When the data nodes starts, we pass the address of the primary node as cmd argument. The data node, registers itself via gRPC to primary node. Then aftear 500 ms, data node sends heartbeat messages to primary node. I have selected gRPC as the communication protocol between the data nodes and the primary node due to its efficiency when compared to HTTP. Data model that each data node sends to primary node via hearbaet is given below.

```
message DataNodeInfo {
  string id = 1;
  google.protobuf.Timestamp last_heart_beat_at = 2;
  repeated Volume volumes = 3;
  string address = 4;
  string http_port = 5;
  string grpc_port = 6;
}

message Volume {
  int32 id = 1;
  string dir = 2;
  int64 used_space = 3;
  int64 free_space = 4;
}
```

Basically, in each heartbeat, the data node sends its current state like volumes, how much space is free in each volume, its network address, http port and grpc port. Based on these data, primary node will decide how to connect to the data node, which volume to write the next object. Note that primary node will store the last heartbeat received from each data node. Once a certain threshold is surpassed without receiving a heartbeat, the data node will be marked as inactive or dead. Consequently, it will be excluded from future write operations, and any attempts to read objects from that data node will result in failure. To implement the heartbeat, we have used the Ticker from the time library. It simply allows us to do a periodic job. When we start the data node, we run separate goroutine which periodically sends the heartbeat to the primary node. See the following code for more details.

```
func (ds *DataServer) StartHeartBeat() {
    log.Infoln("Starting the heartbeat")
    ticker := time.NewTicker(1000 * time.Millisecond)

    go func() {
        for {
            select {
            case ticker := <-ticker.C:
                log.Infoln("Heartbeat at", ticker)
                info := &pb.DataNodeInfo{
                    Id:       ds.ID,
                    Address:  "localhost",
                    Volumes:  asVolumeList(ds.Storage.Volumes),
                    HttpPort: ds.Params.HttpPort,
                    GrpcPort: ds.Params.GRPCPort,
                }
                ds.PrimaryGrpcClient.HeartBeat(info)
            }
        }
    }()
}
```

Id generating logic is implemented on primary node. What we do here is to store the last id used for the object and increase it by one for the next object. After each id request we persist the last value to the local file system. The similar mechanism is implemented for volume id generation as well.

Conclusion

To sum up, heartbeat mechanism is implemented which is important in distributed systems for health checks. To prevent the volume corruptions on multi-thread writing, the locks are introduced on volume.