Faculdade de Ciências da

Universidade do Porto

Assignment 2: Dropbox-ish

Distributed Systems 2017/2018

Authors:
Freddy Vallenilla
Quentin Lebastard

January, 2018

1 Work done by each team member

- Freddy:
  - App server
  - Client
  - Storage controllers: FEC, shard dispatching
- Quentin:
  - Storage controllers: Jgroups
  - Storage pools
  - Deployment to GCP VMs

2 Structure

2.1 Client

CLI with basic authentication using JSON Web Tokens. User is authenticated on connection, and the user's web token is passed to the controller to associate files with users. Commands implemented:

- list: lists all files
- search query: searches filenames for a pattern
- upload/download file: uploads or downloads a file
- delete file: sends a message to delete files

2.2 App server

RESTful API is the entry-point into the system. It effectively proxies traffic to the StorageController and processes some file meta-data. Connections to the StorageController were implemented with low-level sockets for file transfer, and for other messages like file queries we used gRPC.

2.3 Storage controller

Basic structure with some reprentative attributes and operations shown:

2.3.1 Creation

Upon start, the storage controller reads its local and external IP from the local config file (`networkconf.json`). Then, it starts and connects to a JGroups channel.

It then requests the state from the network, which consists of the repartition of the shards in storage pools. That network state will overwrite its local state, and the controller will update its local database accordingly.

Each Storage Controller is started in a different google VM.

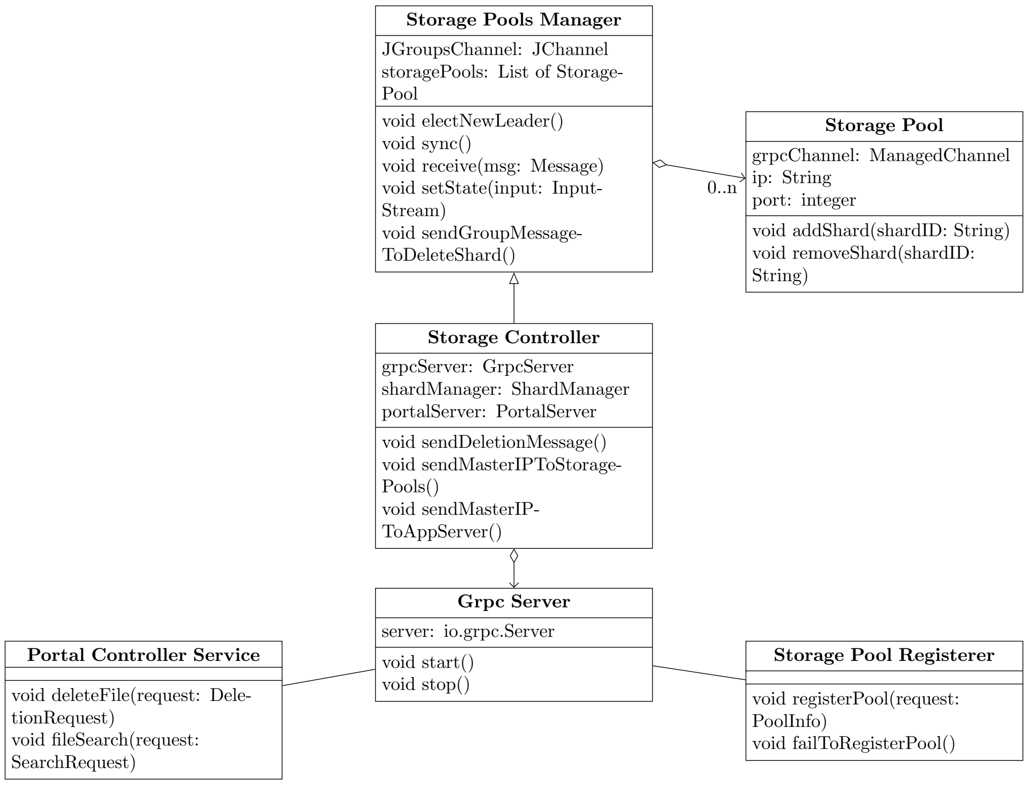
Storage Pools Manager
JGroupsChannel: JChannel
storagePools: List of StoragePool
void electNewLeader()
void sync()
void receive(msg: Message)
void setState(input: InputStream)
void sendGroupMessageToDeleteShard()
Storage Pool
grpcChannel: ManagedChannel
ip: String
port: integer
void addShard(shardID: String)
void removeShard(shardID: String)
0..n
Storage Controller
grpcServer: GrpcServer
shardManager: ShardManager
portalServer: PortalServer
void sendDeletionMessage()
void sendMasterIPToStoragePools()
void sendMasterIPToAppServer()
Grpc Server
server: io.grpc.Server
void start()
void stop()
Portal Controller Service
void deleteFile(request: DeletionRequest)
void fileSearch(request: SearchRequest)
Storage Pool Registerer
void registerPool(request: PoolInfo)
void failToRegisterPool()

Figure 1: The Storage Controller class diagram

2.3.2 Structure

Each storage controller contains a list of the available storage pools, synchronized with the local database.

All the shards from the storage pools are stored in a local SQLite database.

2.3.3 Synchronization

The storage pools and app server only communicate with the master controller, which is the only one running a grpc server.

The master controller stores or deletes the files, and sends message to the other controllers in the jgroups channel to sync them.

The other master controllers update both their database and their runtime data.

2.3.4 Master change

If the master controller disconnects, a new master gets elected. The new master is simply the first controller in the group.

Then, it starts its Grpc server and sends its IP to all the storage pools and to the app server, so they know how to contact it.

All the other storage controllers stop they (eventually running) Grpc servers, so that only the master controller can be contacted by the storage pools and app server.

2.3.5 Network partition

To be able to answer requests, the number of master controllers online needs to be at least half the total number of controllers.

For that, each storage controller stores the maximum number of connected storage controllers it ever saw connected at the same moment.

Example:

Controller	Max controllers connected
1	3
2	3
3	3

Controllers 1 and 2 crash:

Controller	Max controllers connected
1	3

Controller 4 connects:

Controller	Max controllers connected
1	3
4	**2**

At this point, controller 1 is the leader, and it cannot answer requests, because it has still in memory the old architecture of the network. Should the controller 1 fail as well, we will have a totally new network architecture. In this context, the storage controller 4 all alone would then be able to answer requests.

2.3.6 Network unification

When the disconnected controllers connect back to the main group, the overall network state will overwrite their local state and databases.

This solution of overwriting the state was chosen over synchronization for different reasons:

- Simpler to implement (no conflict resolution)
- Don't have to manage time stamps for operation synchronization
- Don't have to log the shard deletions (in the case of synchronization, older deleted shards could come back in existence if not marked as deleted)
- Instead of logging operations on shards in database, we can simply log shards and their location in storage pools

But for that we made a choice on how the storage controllers will behave upon disconnection or failure of some of them, and it's simply: if in the majority group, continue normally - else, do nothing.

2.3.7 Get files from the app server and send them to the storage pools

- The Storage controller gets a file
- It splits it into blocks then shards those with the Reed Solomon algorithm.
- It dispatches shards to storage pools using a simple hashing algorithm (chunk x goes to pool $x\%N_{pools}$)
- It stores the shard location in local database and updates local storage pools list

2.3.8 Communication with the storage pools

The master storage controller communicates with the storage pools via Grpc. The gprc files were generated via protobuf files.

2.3.9 Other elements

Shards Manager: to manage the synchronization of local storage pool info and database info
DBManager: to manage SQLite database operations
Grpc server: to manage grpc interaction with the app server and the storage pools

3 Storage pools

When each storage pool is started, it will first read the master controller's IP adress from its config file. Then, it will try to connect via Grpc to that master controller and register. If it cannot connect, it shuts down.

It then starts a pool of seeders/downloaders, ready to seed or download files on demand.

Each Storage Pool is started in a different google VM.

4 What we didn't have the time to do

Rebalancing after pool crash

- When a storage pool crashes, we do nothing to re-create lost data on the server. However, if enough data shards remain, the file can still be served to the client when requested.
- It would need the master storage controller to re-balance, eg: download every shard of every file affected, re-calculate the shards, and re-dispatch them to a subset of pools.
- All the while keeping track of the location of each shard, old and new, to be able to answer requests from clients

Other

- Multiple user support is currently implemented in the client and app server. Needs to be extended to the Storage Controllers/Pools.
- TLS for GRPC and HTTPS for sockets. We generated certificates for HTTPS connection from client to portal, but did not extend it to the gRPC/tcp messages.

5 Difficulties

Storage controller:

- Managing network partitions: no conflict resolution
- Synchronizing local database and runtime data - done by the ShardsManager class
- Managing shard deletion: chose to simply delete it from database, and have the network state overwrite local state in case of network partition. To sync all the storage controllers, added a *OperationType* to the shard message, which is set to **DEL**

App Server:

- Managing communications between components in two different languages added some complexity.
- A lot of work had to be duplicated. Originally we intended for the erasure coding and decoding to be done in the app-server, which then connects to the pools directly. In the end we ended up extracting all that to the StorageController and it had to be re-written since it is in a different language.
- We had to learn the gRPC API for both Go and Java, which differ significantly.
- The TCP transfer tests were flapping seemingly randomly. When reconstructing the file after an upload and download sometimes the file had a small amount of corrupted data.

6 Technologies used

Trello, Git, SQLite, Google Compute Engine, gRPC, JGroups, Google Protobuf, Golang