Dataverse

Project Report

CMPE 275

Team Members
Priya Khadke(013826631)
Harshada Jivane(013855751)
Manasa Hari(013855075)
Ankit Thanekar(013729911)
Shailesh Nayak(014529151)

Table of Contents

Introduction

This project is a scalable, decentralized, robust, heterogeneous file storage solution that ensures that multiple servers can inter-operate to store and retrieve files in a distributed and fault-tolerant way.

Problem Statement:

Our core goal is to build a dataverse architecture which supports 3 main capabilities:
1) Massive registration
2) Support to all file extensions
3) Efficient storage and search capabilities

Brainstorming

At first, looking at the problem we arrived at an intuition of building a single server and then scale it horizontally. In this process of brainstorming, the key considerations were data size, data replication factor, and loosely coupled architecture. We want a reliant bookkeeper with respect to file storage. We built our entire architecture keeping the single responsibility principle in mind. And that's why we separated the bookkeeper from the webserver. We maintained the bookkeeper in the service registry. We arrived at an optimal solution after multiple iterations in the development stage.

Use Case

We chose basic file upload and download as our use case.

The system supports the following design aspects:

- Scalability
- Fault Tolerance
- Parallel retrieval
- Caching
- Consensus
- Single Responsibility principle

Users of this system can perform the following operations:

- File Upload (txt,jpeg,png,mp3,mp4) [audio,video and text]
- File Download

Admin of this system can perform the following operations:

- Heartbeat Status
- Leadership Status

What we concluded after brainstorming:

In our optimal solution, we used 3 main components. 1) An application server 2) Service registry and 3) Actual gRPC or streaming server.

We successfully achieved two main optimizations:
1) Caching
In this method, we developed a reliable way of first checking in the gRPC server for data. If not found there, it will then search for data in the s3 persistence storage.

2) Parallel execution (thread model)
Both uploading and downloading of files are executed in parallel thread model. Each file is broken down into different chunks. Each chunk will then be associated with a thread. And these threads are executed in parallel. This enabled an efficient parallel streaming mechanism.

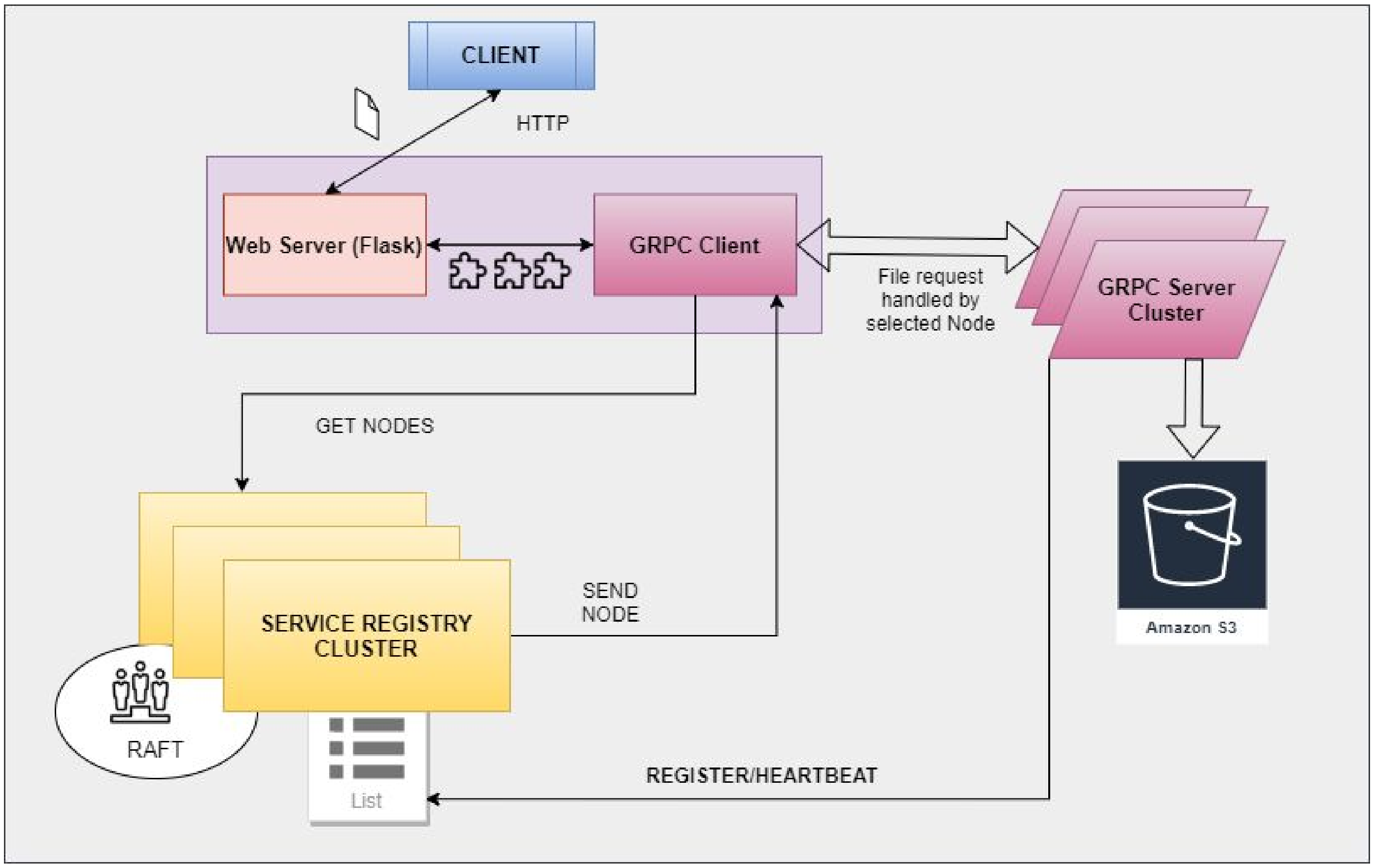
Algorithm flow

We built three main components in this system. We also enforced loosely coupled design.

They are 1) Application server 2) Service registry and 3) The actual gRPC server.
We maintained three server instances, each of different type and location. We used Amazon s3 for the persistence storage of data. When we upload a file, the file is broken down into several chunks based on the file size. These chunks are registered corresponding to the filename in the service registry. Based on the MD5 mapping, the chunks are then uploaded to the respective server. The default chunk size is 30 MB. So, if a file of 150MB is uploaded, it is divided into 150/30=5 chunks. Now, these chunks form an array mapped to the filename in the service registry. Each chunk has its sequence number. Based on the sequence number, we know the order of chunks.

The same algorithm above works for downloading files too. While uploading and downloading, we wait till all chunks are successfully collected before updating the registry or allowing it to be downloaded fully.

System Design



Components:

1) Client:
    - Uploads and downloads file over HTTP
2) Web Server
    - Handles HTTP Client requests.
    - Gets 'best' server from Service Registry
3) GRPC Client
    - Splits file into small chunks and send over to the selected server
    - Part of the Web Server instance
4) GRPC Server
    - Worker process
    - Caching
    - Store in S3
5) Service Registry
    - Highly available (RAFT)
    - Consistent Hashing to a selected server

    - Monitoring
    - Bookkeeping
6) Amazon S3 Bucket
    - Centralized file storage
    - Help replication that avoids a single point of failure

Implementation

Scaling

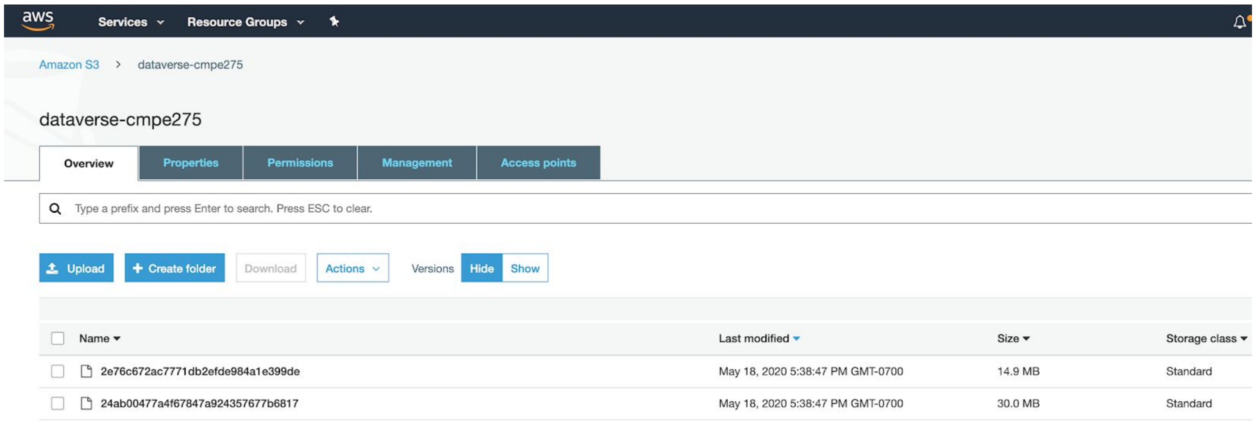
- We try to maximize our ability to scale horizontally.
- Web(Application) Server is decoupled from Service Registry allowing it to scale freely.
- Service Registry is a cluster(3-7) of highly available servers that are only responsible for load balancing and bookkeeping.
- The GRPC Server(Worker) cluster can be scaled independently.

Load Balancing

- Using only the hash of the id of the data you can determine exactly where that data should be.
- This mapping of hashes to locations is usually termed a “ring”- Hash Ring
- We will have a separately numbered ids, so data would be balanced between the two machines based on the load size.
- Service Registry is responsible for providing the GRPC client with the server node which needs to forward the requests from Web Server.
- Consistent Hashing is implemented to get the appropriate server from the list ensuring sticky sessions with active servers.

Sharding

- Each data file is partitioned into several parts called chunks.
- Each chunk is handled by a different server and uploaded to the central file system(S3) at the same time, caching it for further download requests.

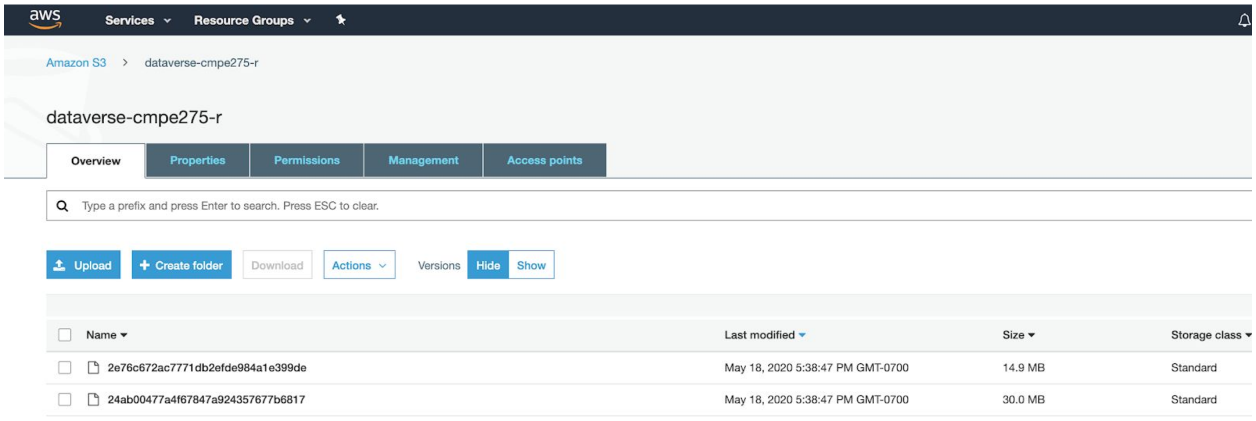


dataverse-cmpe275

Name	Last modified	Size	Storage class
2e76c672ac7771db2efde984a1e399de	May 18, 2020 5:38:47 PM GMT-0700	14.9 MB	Standard
24ab00477a4f67847a924357677b6817	May 18, 2020 5:38:47 PM GMT-0700	30.0 MB	Standard

Fault Tolerance

- Service Registry is aimed at being highly faulted tolerant by the use of the RAFT Consensus algorithm for persistent storage.
- GRPC Server cluster has the capability to scale horizontally which makes this cluster fault-tolerant (unless all fail, of course)
- S3 replicates these chunks internally to provide resilience to failures or data loss.
- The image below shows the replicated chunks from the dataverse-cmpe275 bucket to the dataverse-cmpe275-r bucket.



dataverse-cmpe275-r

Name	Last modified	Size	Storage class
2e76c672ac7771db2efde984a1e399de	May 18, 2020 5:38:47 PM GMT-0700	14.9 MB	Standard
24ab00477a4f67847a924357677b6817	May 18, 2020 5:38:47 PM GMT-0700	30.0 MB	Standard

Consensus using RAFT

- The consensus is a fundamental problem in fault-tolerant distributed systems. Consensus involves multiple servers agreeing on values. Once they reach a decision on some value, that decision is final.
- The raft is integrated using the PySynObj library. This library provides us a way to have replicated persistent data structures to store files and server information across the cluster.
- Leader election is mainly done internally to commit these updates.

Caching - Optimization

- GRPC servers perform in-memory caching of file chunks upon the first download request.
- Consistent hashing provides us sticky sessions to these servers.
- On a cache miss, they fetch the requested chunk from central file storage.

Performance Validation

Setup

1. EC2 T2 Micro Ubuntu server
2. 3 Instances EC2 T2 micro
3. Testing with s3 upload

Test Cases

1. (file stored as chunks)Upload file test with diff file sizes and response times
2. (file stored as chunks) Download File with diff file sizes and response times
3. (file stored as a single unit) Upload file test and response times
4. (file stored as a single unit) Download File and response times
5. Testing with single chunk upload

Test Results

1. Single chunk Upload Request 86.17s
2. Single chunk Download Request 26.7s

1 GRPC server with s3 upload and download

a. 22MB Upload 1 Chunk 15.9s download time 4.76 s
b. 62MB Upload 2 Chunk 37.17s download time 13.36s
c. 169MB Upload 6 Chunks 86.7s download time 29.4 s

2 GRPC server with s3 upload and download

a. 62MB Upload 2 Chunks 38.92s download time 10.15 s
b. 169MB Upload 6 Chunks 79.4s download time 21.64 s

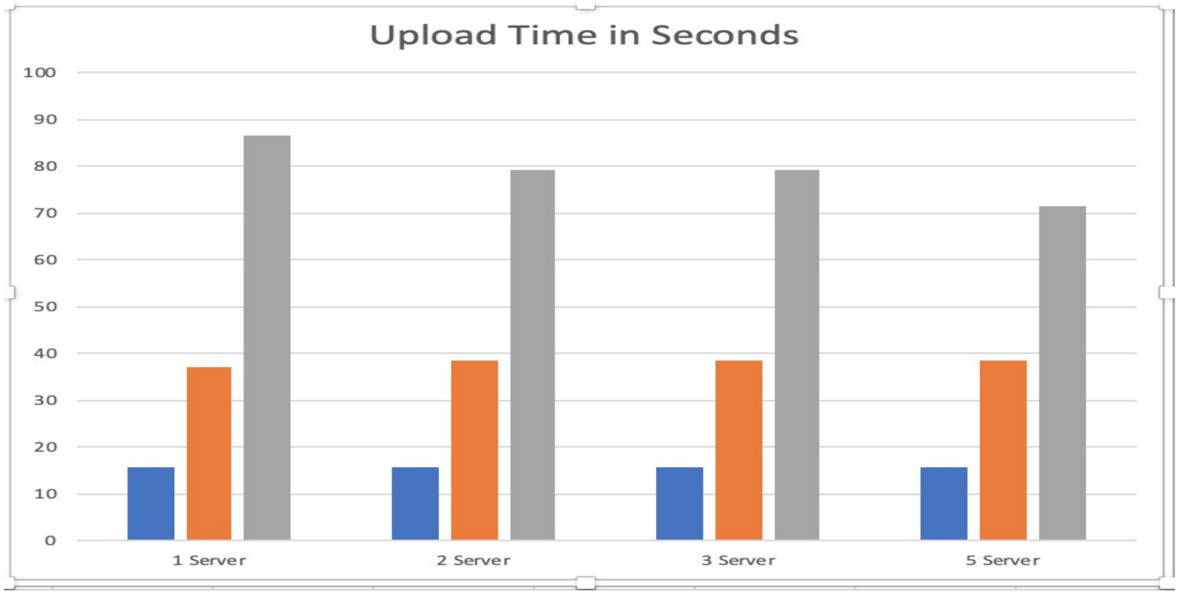
3 GRPC server with s3 upload and download

a. 169MB Upload 6 Chunks 79.32s download time 17.76 s

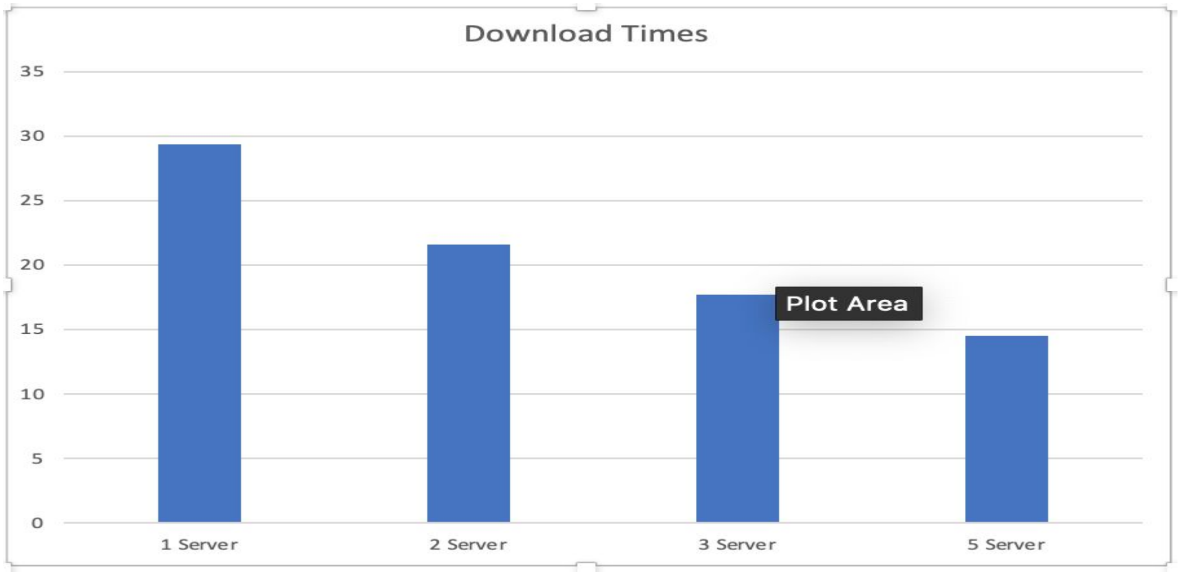
5 GRPC server with s3 upload and download

b. 169MB Upload 6 Chunks 71.4s download time 15.4 s

Test Graphs



Upload Time in Seconds
100
90
80
70
60
50
40
30
20
10
0
1 Server
2 Server
3 Server
5 Server



Download Times
35
30
25
20
15
10
5
0
1 Server
2 Server
3 Server
5 Server
Plot Area

Responsibilities (Task Division)

Team Member	Responsibilities
Priya	Architecture and design research, GRPC client-server file chunking, and storage. Service registry consistent hashing integration. Enhancement around parallel chunks retrieval and caching lookup.supported RAFT research.
Shailesh	Architecture and design research, RAFT research and design, service registry API for fetching latest maps, reporting
Harshada	Architecture and design research, Configuring Amazon S3 Buckets and Access Key for application to reach S3 bucket. Fault Tolerance by assuring data availability and replication, bug fixes, reporting
Ankit	Architecture and design research, Dataverse cloud deployment. Performance testing and bug fixes.service registry APIs for register and heartbeat
Manasa	Architecture and design research, Added validations around upload and download, Dataverse monitoring application development, Front end development.

Conclusion

A distributed and scalable system for data storage and retrieval was implemented.

Future Scope

- gRPC servers & Web Servers can be scaled horizontally.
- Efficient multithreading and validation around a number of threads.
- More robust architecture for security.
- The more efficient approach can be adopted by caching the file chunks on the streaming server by the in-memory store. In the current implementation, it is on the file system of a streaming server.

References

1. https://ops.tips/blog/sending-files-via-grpc/
2. https://grpc.io/docs/tutorials/basic/python/
3. https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

4. https://raft.github.io/
5. https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-examples.html
6. https://docs.aws.amazon.com/AmazonS3/latest/gsg/CreatingABucket.html
7. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html#Using_CreateAccessKey

Git Repo: https://github.com/priyakhadke15/dataverseproject

Demo Video Link:
https://www.youtube.com/watch?v=dNCXMMdeFc4&feature=youtu.be