Simple Object Storage - Interim Report

Nijad Huseynov

July 2023

1 Introduction

In this research class, I am implementing a distributed object storage for small files based on Facebook's Haystack paper. The project aims to address the challenges associated with efficiently storing and fetching a large volume of small objects in a distributed architecture.

Traditional file systems often is not a good fit for large volumes of small files due to overhead of metadata. Since each read requires 3 disk IO request to the storage layer.

In this project, I am aiming to improve the overall performance of the storage system by minimizing disk IO operations. This will enable to handle large volumes of small files and provide faster access to objects in storage.

Key objectives of the project are following

- Designing and implementing a distributed object storage based on the ideas from the Facebook haystack paper
- Optimizing the object storage for large volumes of small files
- Minimizing the metadata operations on small files, thus improving the read performance of the system

2 Key ideas

The basic unit of the data will be **needle**. Needle represents small object.

```
type Needle struct {
    TotalSize uint32
    Id        uint32
    NameSize  uint32
    Name      []byte
    MimeSize  uint32
    Mime      []byte
    DataSize  uint32
    Data      []byte
    Checksum  uint32
}
```

The key idea in the implementation is the merge small objects into one large object and store them as single file on local file system. This in itself does not improve the read performance of the systems. Because, each time to retrieve an object, we have to walk a large file and find the object with the given id which is quite inefficient. To prevent that, I will keep an in memory mapping of the object id to offset so that, when there is a request for an object we can easily fetch it from the memory using the offset. Previous two ideas are the keystones of the project and the whole implementation will be based on that.

3 Challenges

Data corruption Delete operation Garbage collection

4 Upcoming task

The tasks in my project backlog are shown below.

- Add volume layer to data node - Currently, there is no abstraction between needle and storage. Adding a volume layer will help me provide a better lock mechanism. The relationship will be as follows: Each disk is storage. Storage will have volumes. Volumes (volume simply refers to large files in this project's context) will have needles.
- Implement lock mechanism on volume layer so that multiple thread does not corrupt the data - Currently multiple write may corrupt the volume
- Implement an in-memory map of object offsets for fast lookups - This is the one of the most important part of the implementation to reduce disk IO
- Implement heartbeat mechanism (probably use gRPC) - Hearbeat on data nodes will send volume statistics to primary server. One use case of that data, primary server will decide where to write the next object.
- Implement a mechanism that reads all the data in volumes and initiates the in-memory map.
- Add flag byte to needle
- Implement checksum to detect object corruptions
- Implement configs. The user must be able to start the data node with different config parameters.
- Implement an algorithm that decides where to write the next object - This will be implemented on the primary node.
- Implement endpoints for primary server.
  - Endpoint to get an object id and address of the data note to write the data
  - Endpoint to get the address of the data node where the object can be fetched by object id
- Implement id generator for objects(snowflake can be an option)
- Implement a gRPC server that receives heartbeats from the data nodes.
- Test if the data is uniformly written to the data nodes
- Write scripts to test the integrity of the systems (check if the object is corrupted after being stored in the systems).
- Implement/perform different workload test (described in report 3) for performance evaluation (other tools can be utilized as well).

5 Conclusion