Simple Object Storage - Interim Report

Nijad Huseynov

July 2023

1 Introduction

In this research class, we are implementing a distributed object storage for small objects 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 about three disk IO request to the storage medium.

In this project, we are 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

The rest of this report is organized as follows. In section 2, we present current architecture of the system and key ideas in the implementation. In addition it shows how the write/read/delete/update path is designed. Section 3 presents current challenges and the possible solutions to those problems. Section 4 presents the tasks that needs to be done in the project backlog. Finally, section 5 concludes the report.

2 Project Overview

2.1 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, we 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.

2.2 Architecture

2.2.1 Cluster components

The current architecture ("Fig. 1") has one primary node and many data nodes. Primary node will be responsible for managing the the data nodes and data nodes will be responsible for storing the actual object.

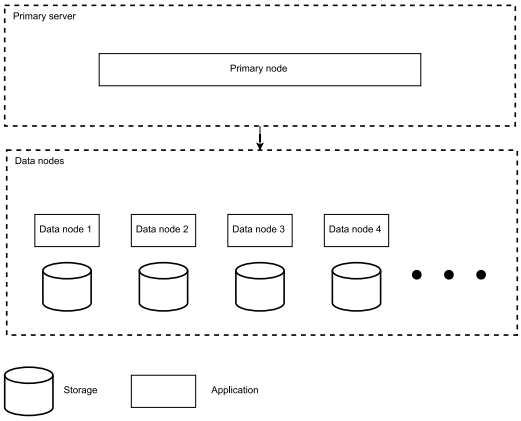

Figure 1: Current architecture of the system

In the first draft architecture of the project proposal, we proposed two kinds of nodes: metadata nodes and data nodes. The metadata node would store the metadata of the objects, and the data node would store the actual content of the objects. One problem with this architecture is that maintaining consistency across the cluster can be hard. The second problem is scalability. Because to make the system horizontally scalable, we have to have more than one metadata server, and this complicates the implementation.

The new proposed architecture does not separate metadata from the content of the object. This fixes the scalability problem. If we need to scale, we will simply add a new data node to the cluster. And the primary node (previously the metadata node) will just manage the data nodes and rarely become a bottleneck in the architecture.

2.2.2 Data node components

Fig. 2 shows the current components of the data node application. On top of the stack, we have api handler which will be responsible for api calls. Then the request will be passed to the server layer. The server layer will be responsible for translating the request to the form storage layer understands. The storage layer represents the physical storage and will have many volumes. Each volume will store many small objects (needles).

2.3 Core operations on object

2.3.1 Write path

To start a write, the client first ask the primary node where to write the data. Based on volume information, primary node will decide which data node to write the next object. Additionally, the primary node has to create a unique object ID for each object ("Fig. 3").

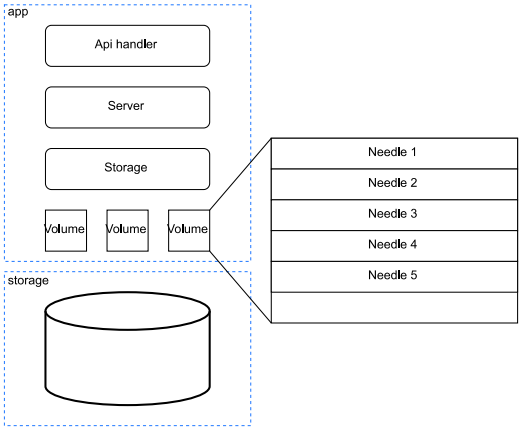

Figure 2: Current components of the data node

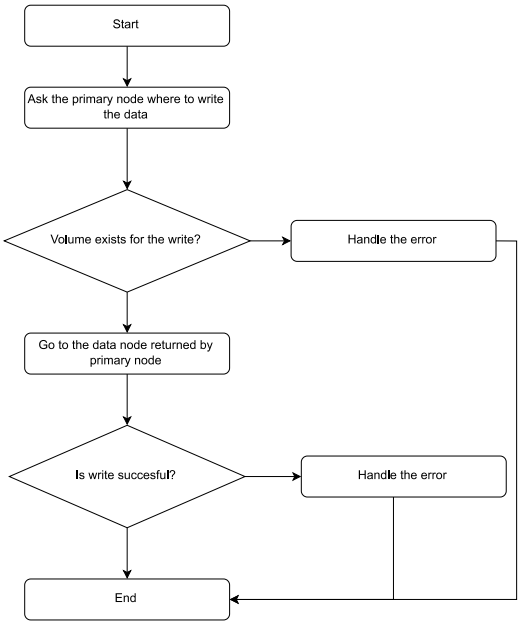

Figure 3: Write path

2.3.2 Read path

To read an object, client sends a request with an object id to primary node. Based on the id, primary node finds which data node has the object. Then client goes to the data node with the object and requests the read operation (“Fig. 4”). Data node will go check its in memory mapping of the object id to offset. After knowing which offset the object is, it will read starting from the offset and return the result.

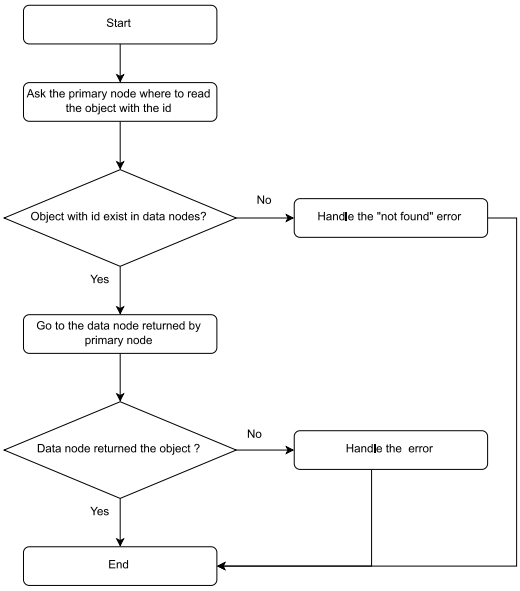

Figure 4: Read path

2.3.3 Delete path

To delete an object, client sends a request with an object id to primary node. Based on the id, primary node finds which data node has the object. Then client goes to the data node with the object and requests the delete operation (“Fig. 5”). After the delete operation, data node will mark the object as deleted and later on it will be garbage collected.

2.3.4 Update path

To update an object, the client will simply write the object with the same id in the volume on data node. Previously written object will be ignored during the read operation and garbage collected later.

3 Challenges

One of the problems in the implementation is that we have to make sure that, we are reading what we wrote to the storage. If there is a corruption, the system should detect it. To do that, we will implement cyclic redundancy check.

Another challenge is the delete operation. Since we are appending the objects sequentially to the end of the volume file, we can not efficiently delete the objects in place. To speed up the delete operation, we will just mark the object as deleted and during the read operation we will check if the object is deleted or not. This introduces storage problem. If there is a lot of delete operations on a volume we will waste disk storage and additionally the volume file will be fragmented. To address this issue, we have to make a garbage collection.

4 Upcoming task

The tasks in our project backlog are shown below.

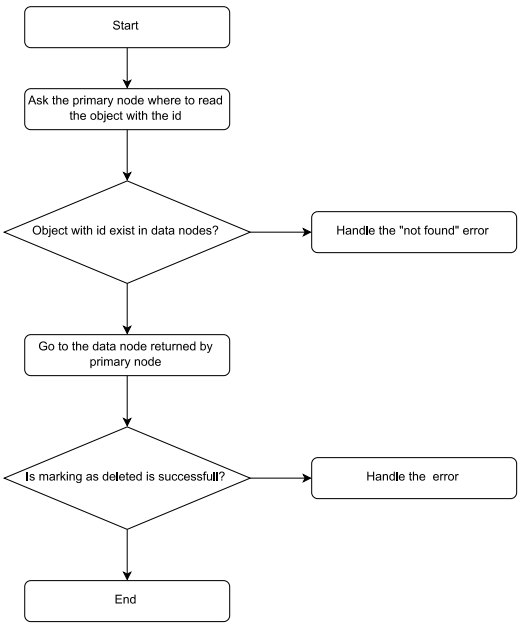

Figure 5: Delete path

- Add volume layer to data node - Currently, there is no abstraction between needle and storage. Adding a volume layer will help us 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).

Note that this is not an exhaustive list, we may add extra tasks during the implementation.

5 Conclusion

In summary, this report presents the key updates in the project implementation, challenges during the implementation as well as their possible solutions, and a roadmap for the upcoming weeks.