Simple Object Storage

Distributed object storage optimized for small files

Nijad Huseynov
Advisors: Stephen H. Kaisler, Jamaladdin Hasanov
SEAS GW University, SITE ADA University

3 August 2023

Outline

Project Objectives

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

Features

The system is optimized for small objects and implemented for horizontal scalability

- ID generation for objects
- Volume manager. It is rule based periodic job that checks the statistics of the volumes from data node
- Object read and write
- Heartbeat mechanism
- Object corruption detection on read

Key ideas for optimizing for small files

In memory mapping of object id to the offset

Hashmap data structure is used.
Key is the object id and the value is offset of the object in the volume file.

Merge small files into one large file

Lot of small files will have lots of metadata in local filesystem and it will be hard to cache those metadata. Thus we create one large file and append the object.

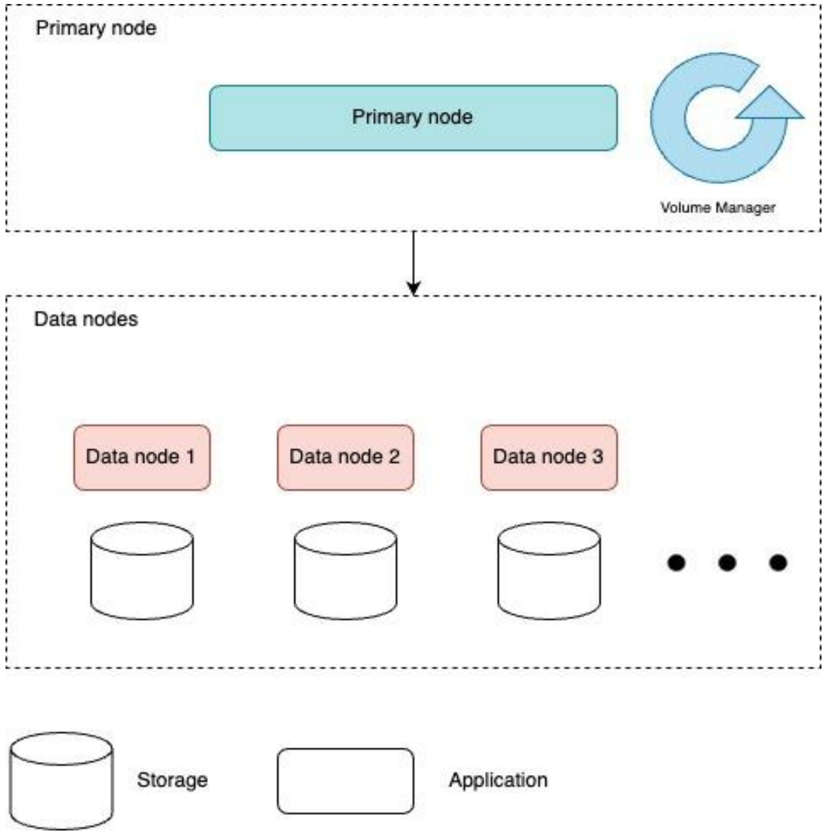
Keeping file descriptor open for each volume to reduce the disk IO requests to O(1)

Since we will have one large file, it will be easy to cache metadata of the file in ram, thus reducing the disk IO to 1

System design

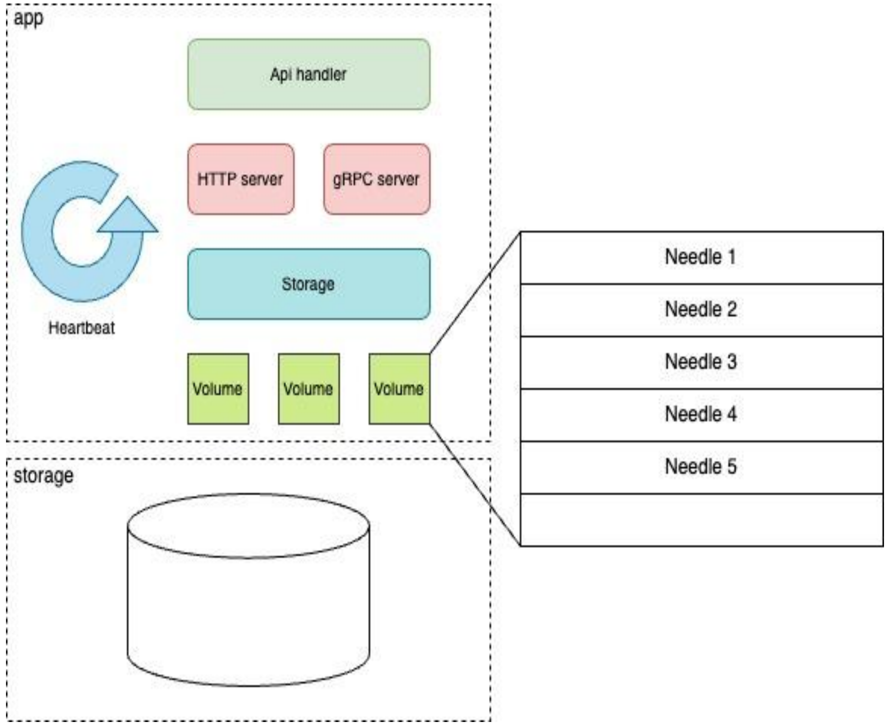
- Primary node is responsible for managing volumes
    - Volume manager
    - ID generator
    - Cluster info

- Data node is responsible for storing data
    - Stores the objects in volumes as “needles”
    - Read and write apis for objects
    - Heartbeat process to primary node



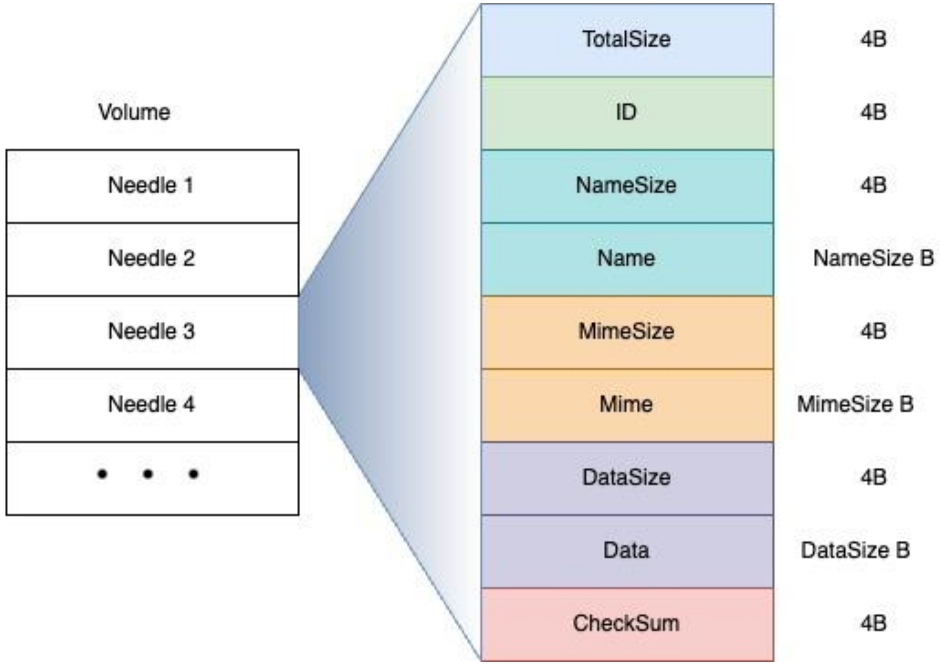
Data node components

- **Api handler** is responsible for the api calls. In this layer, there will be no logic apart from some input sanity checks. It will basically parse request params and pass it to the server layer
- **Server layer** will be responsible for the translation of the incoming requests to the storage layer like constructing the needle to the format storage layer understands and so on.
- **Storage** layer is responsible for the management of the volumes.
- **Volume** will contain some number of small objects(needles).



Needle - unit of data in the system

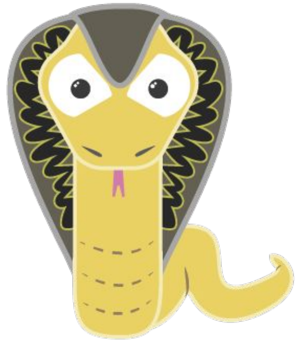
```
type Needle struct {
        TotalSize    uint32 # Total size of Needle
        Id      uint32 # Id of the writtent obj
        NameSize     uint32 # Number of bytes in name
        Name         []byte # Obj name
        MimeSize     uint32 # Obj mime size
        Mime         []byte # Obj mime type name
        DataSize     uint32 # Obj content size
        Data         []byte # Obj's content in bytes
        Checksum     uint32 # Checksum of the obj
}
```



Implementation

- Golang is used for the implementation
  - Simple
  - Fast
  - Concurrent
  - Portable

- Following open source golang libraries are used
  - Echo - for the http server
  - gRPC Go - for the gRPC server
  - Cobra - for the command line interface implementation
  - Logrus - for the logging

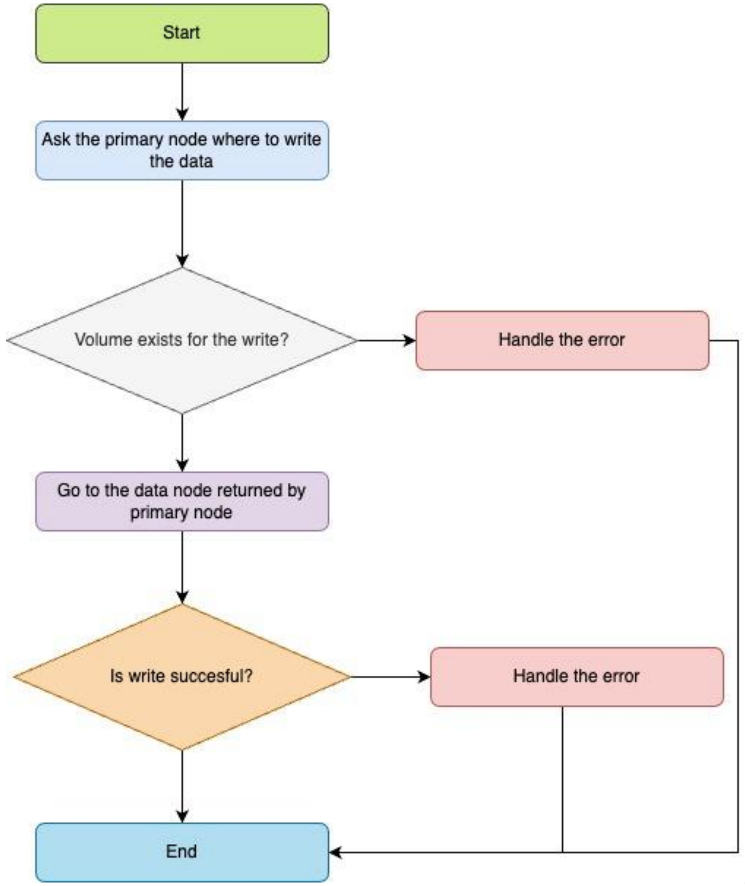


Object write flow

Step 1. Client asks the location to write the object from the primary node

Step 2. Primary node returns the address of the data node which object can be written

Step3. Client writes the object to the given data node.

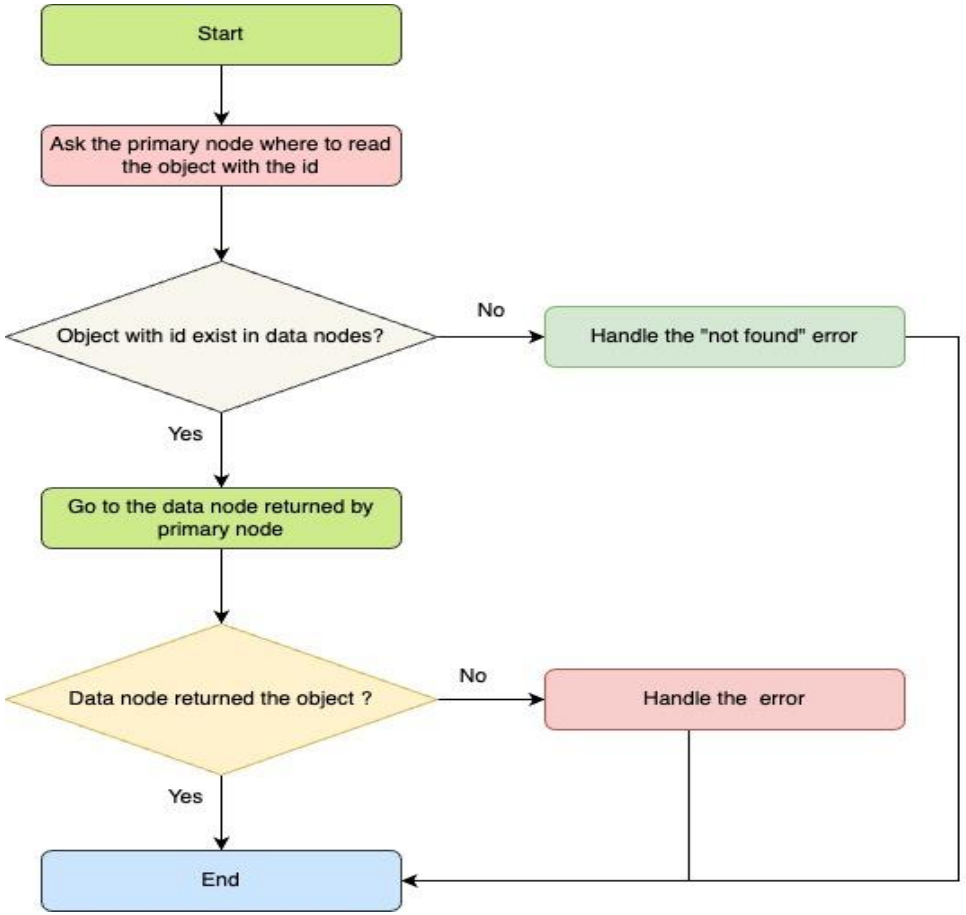

Start
Ask the primary node where to write the data
Volume exists for the write?
Handle the error
Go to the data node returned by primary node
Is write succesful?
Handle the error
End

Object read flow

Step 1. Client asks the location of the object from the primary node

Step 2. Primary node returns the address of the data node which has the object

Step 3. Client requests the object from the data node.

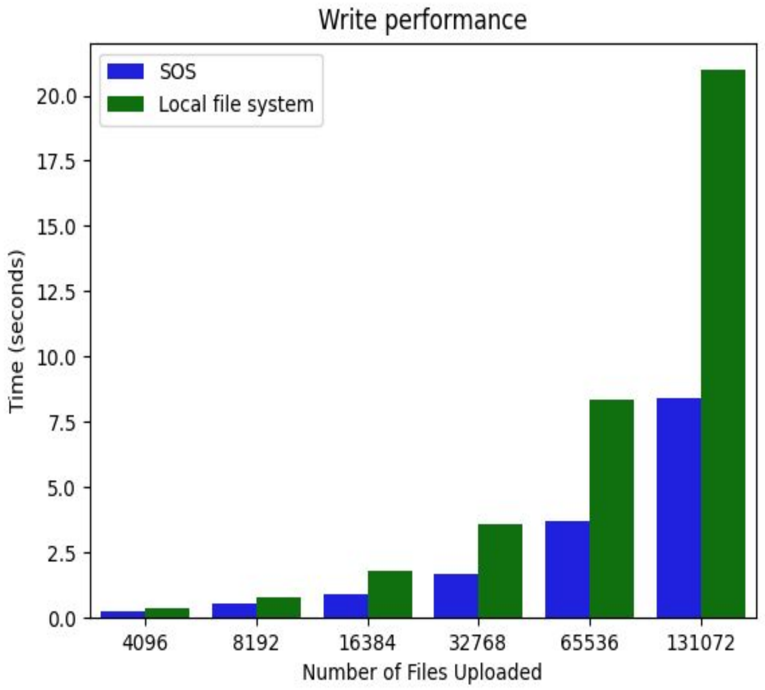

Start
Ask the primary node where to read the object with the id
Object with id exist in data nodes?
No
Handle the "not found" error
Yes
Go to the data node returned by primary node
Data node returned the object ?
No
Handle the error
Yes
End

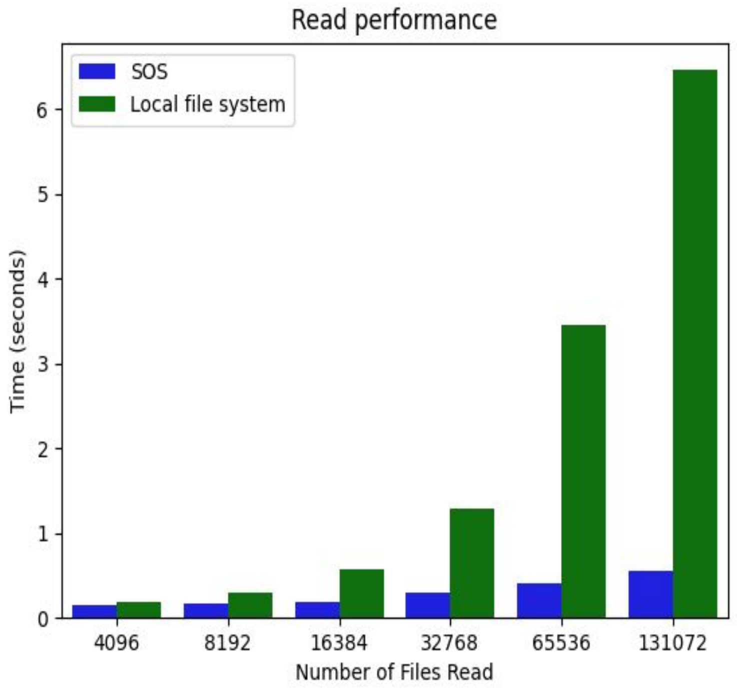
Evaluation

- Setup
  - Apple M1 chip, 8 cores
  - 8GB RAM
  - 256 SSD with APFS

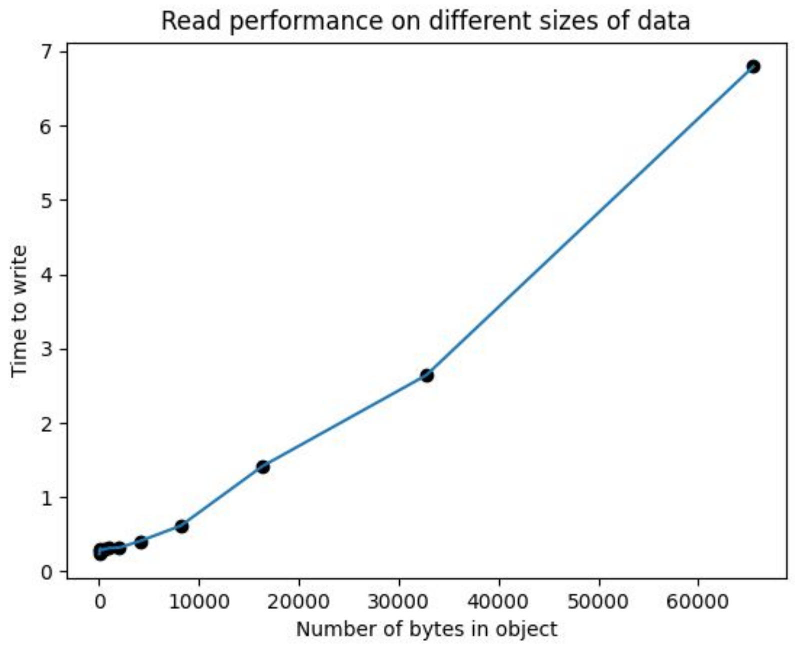
- Workloads
  - Read/Write performance of the systems against local file system with 1KB files
  - Read/Write performance of the system with different bytes of data

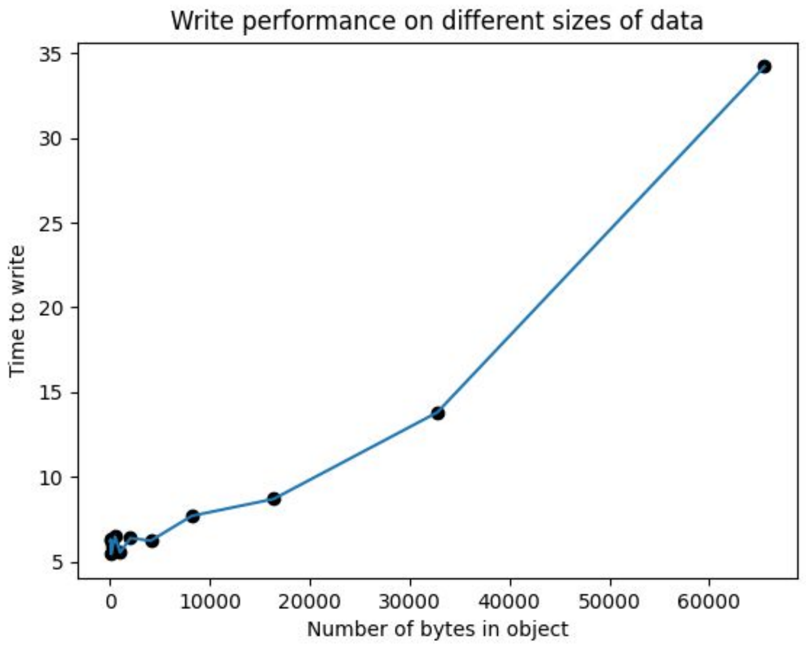
Results: Local File System vs SOS on 1KB data





Results: Read/Write performance on different bytes of data





Number of files in each test case: 100,000

Data size: 32 B, 64 B, 128 B, 256 B, 512 B, 1024 B, 2048 B, 4096 B, 8192 B, 16384 B, 32768 B, 65536 B

Conclusion

Distributed Object storage is implemented for small files

It is horizontally scalable

Heartbeat mechanism is implemented as an healthcheck

Evaluation shows that system performs well on small sized data compared to local file system

Future work

- Object deletes/updates
- Garbage collection
- Index for the object id to offset mapping
- Batched writes to take advantage of the HDDs

Any Questions?

Backup

Starting the system

To start the primary node

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

To start the data node

```
./sos data --vol_dir="tmp/node1"

           --primary_node="localhost:1234"

           --port="8081"

           --grpc_port="1235"

            --node_id="1"
```

APIs

Primary Node

GET **/primary/volume**

GET **/primary/search?id=12**

GET **/primary/cluster-info**

Data Node

POST **/data/:fid**

GET **/data/:fid**