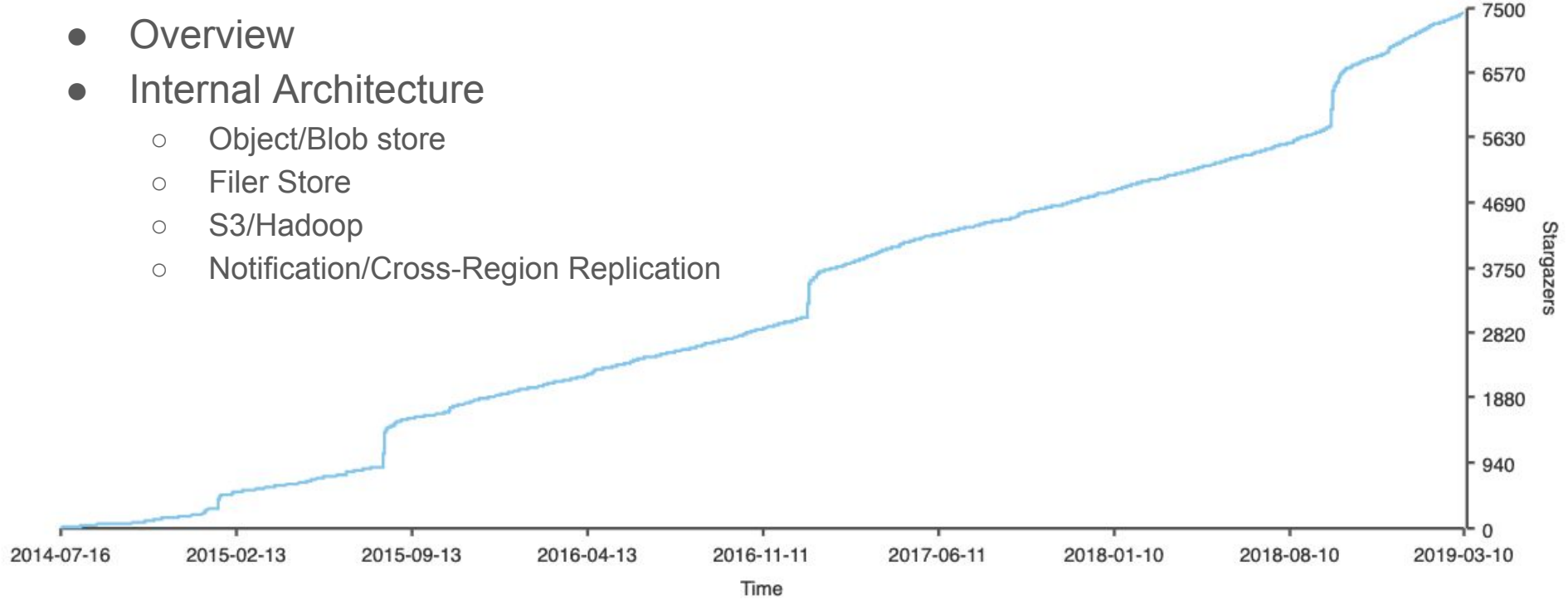
SeaweedFS Intro

2019.3

chris.lu@gmail.com

Stargazers over time

- Overview
- Internal Architecture
  - Object/Blob store
  - Filer Store
  - S3/Hadoop
  - Notification/Cross-Region Replication



SeaweedFS Intro

Start Master Server

```
> ./weed master
```

Start Volume Servers

```
> weed volume -dir="/tmp/data1" -max=5  -mserver="localhost:9333" -port=8080 &
> weed volume -dir="/tmp/data2" -max=10 -mserver="localhost:9333" -port=8081 &
```

Write File

To upload a file: first, send a HTTP POST, PUT, or GET request to `/dir/assign` to get an `fid` and a volume server url:

```
> curl http://localhost:9333/dir/assign
{"count":1,"fid":"3,01637037d6","url":"127.0.0.1:8080","publicUrl":"localhost:8080"}
```

Second, to store the file content, send a HTTP multi-part POST request to `url + '/' + fid` from the response:

```
> curl -F file=@/home/chris/myphoto.jpg http://127.0.0.1:8080/3,01637037d6
{"size": 43234}
```

To update, send another POST request with updated file content.

For deletion, send an HTTP DELETE request to the same `url + '/' + fid` URL:

```
> curl -X DELETE http://127.0.0.1:8080/3,01637037d6
```

Overview: What is special?

- Distributed
- Handles large and small files
- Optimized for large amount of small files
- Random access any file
- Low-latency access any file
- Parallel processing

Overview: APIs

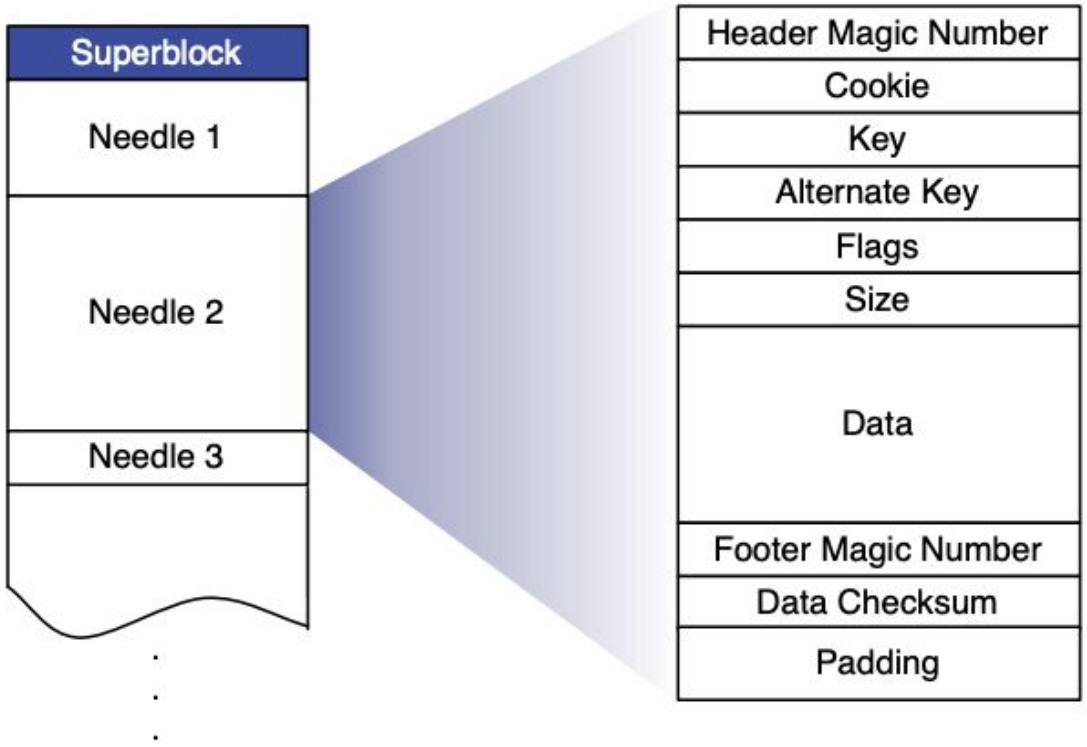
- REST API for object storage
- REST/gRPC API for file system storage
- Hadoop Compatible
- FUSE client to mount file system locally
- S3 API

Architecture

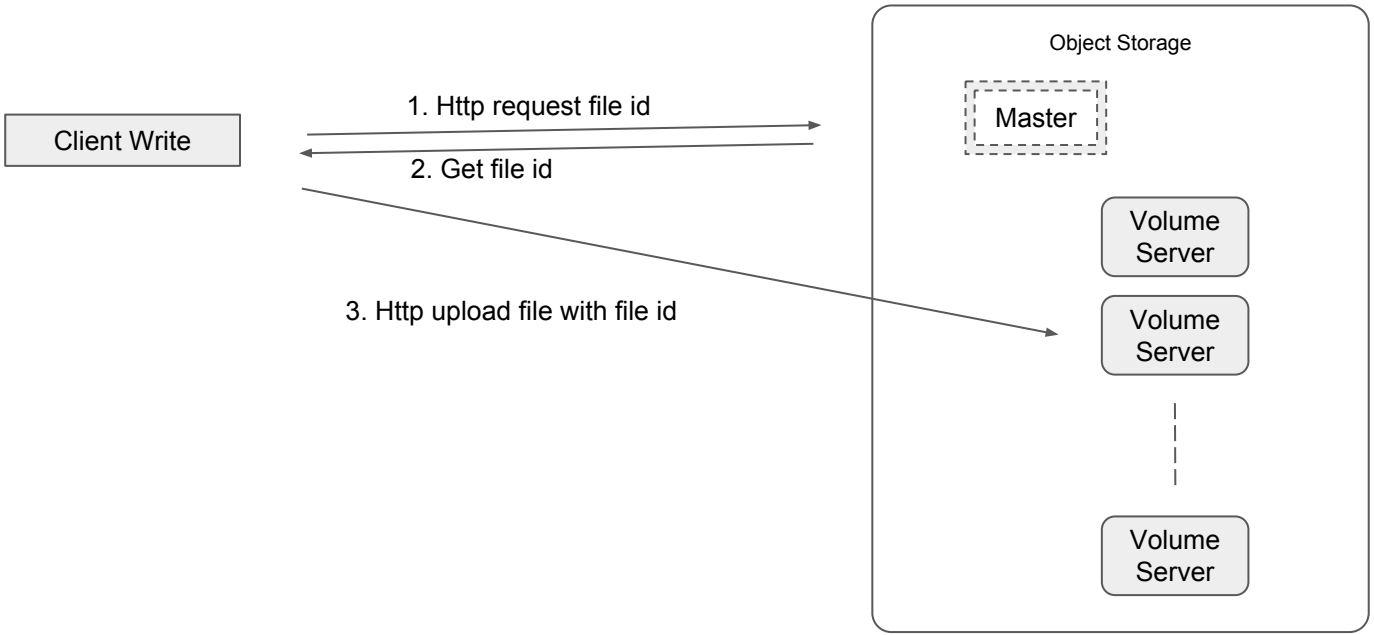
- Object Storage
- File Storage
- Interface/Client Layer

Volume Store

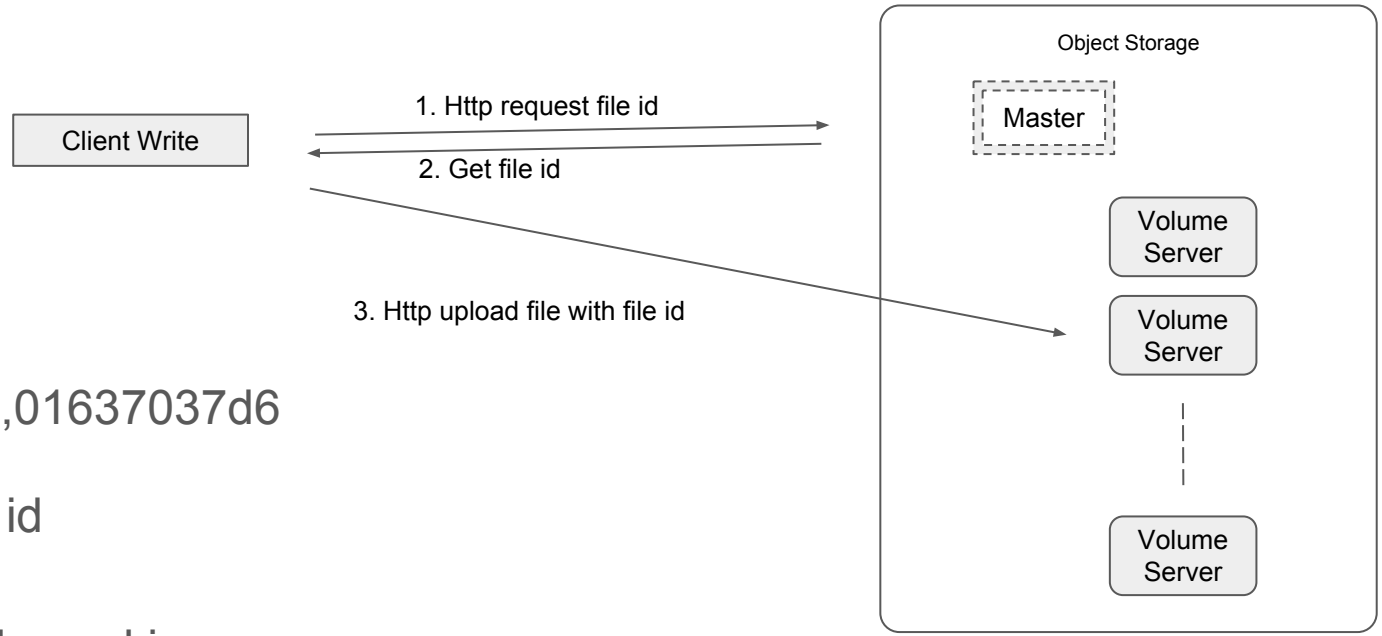
- Based on [Facebook Haystack paper]



Object Storage



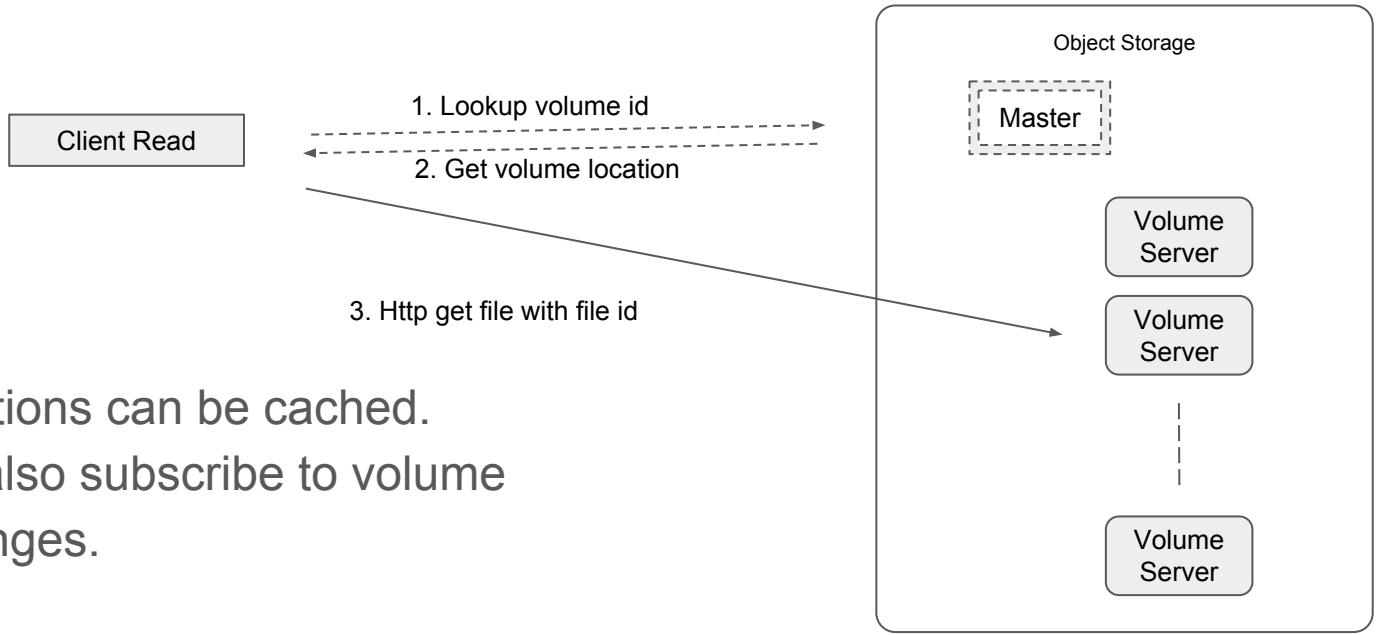
Object Storage



Example file id, 3,01637037d6

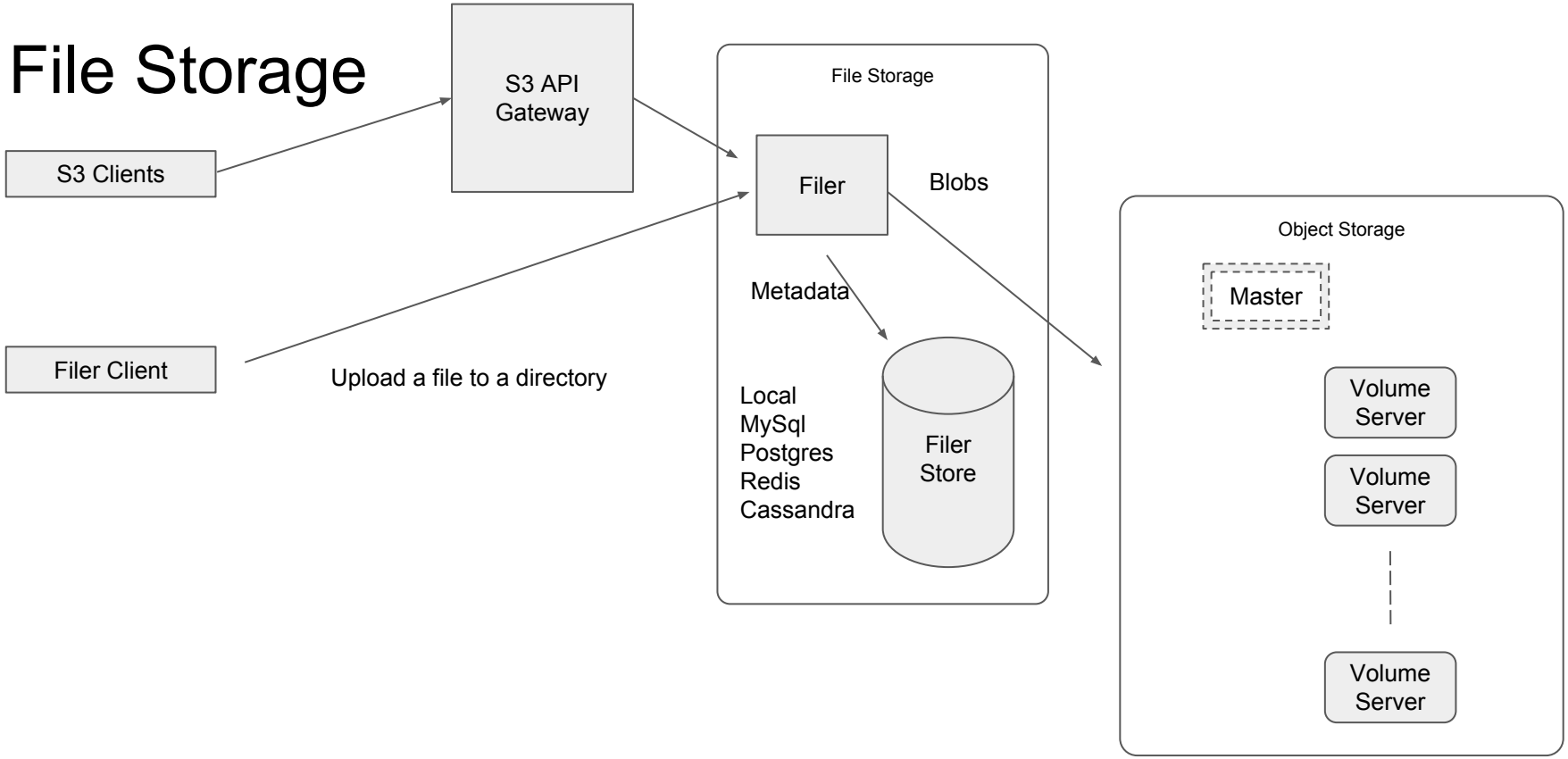
- 3 : a volume id
- 01: file key
- 637037d6: file cookie

Object Storage



- Volume locations can be cached.
- Clients can also subscribe to volume location changes.

File Storage



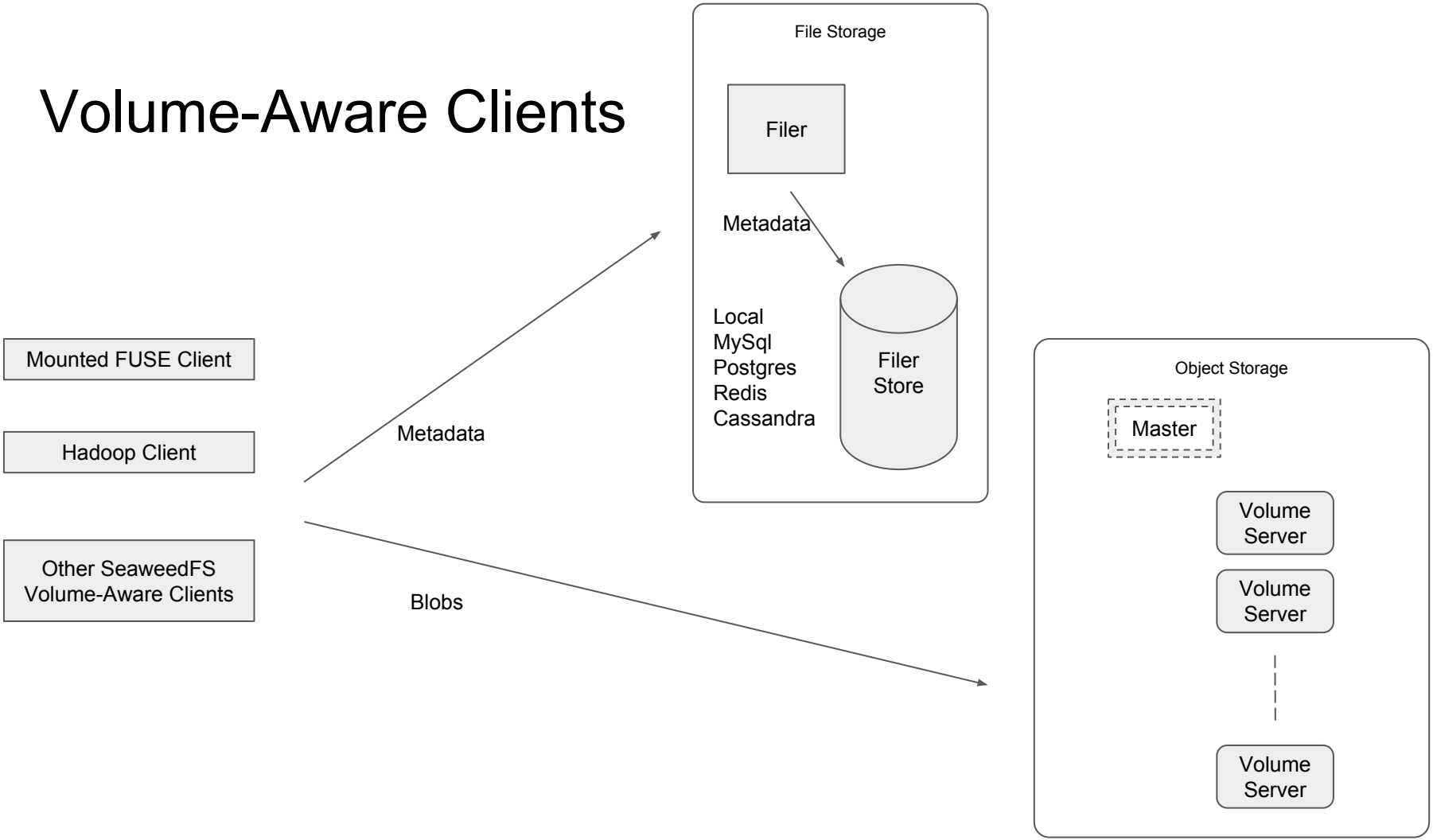
Filer Store Data Layout

/a/b/c/	Attr	
/a/b/c/def.txt	Attr	FileChunks

```
type Attr struct {
    Mtime          time.Time
    Crtime         time.Time
    Mode           os.FileMode
    Uid            uint32
    Gid            uint32
    Mime           string
    Replication    string
    Collection     string
    TtlSec         int32
    UserName       string
    GroupNames     []string
    SymlinkTarget string
}
```

```
type FileChunk struct {
    FileId       string
    Offset       int64
    Size         uint64
    Mtime        int64
    ETag         string
    SourceFileId string
}
```

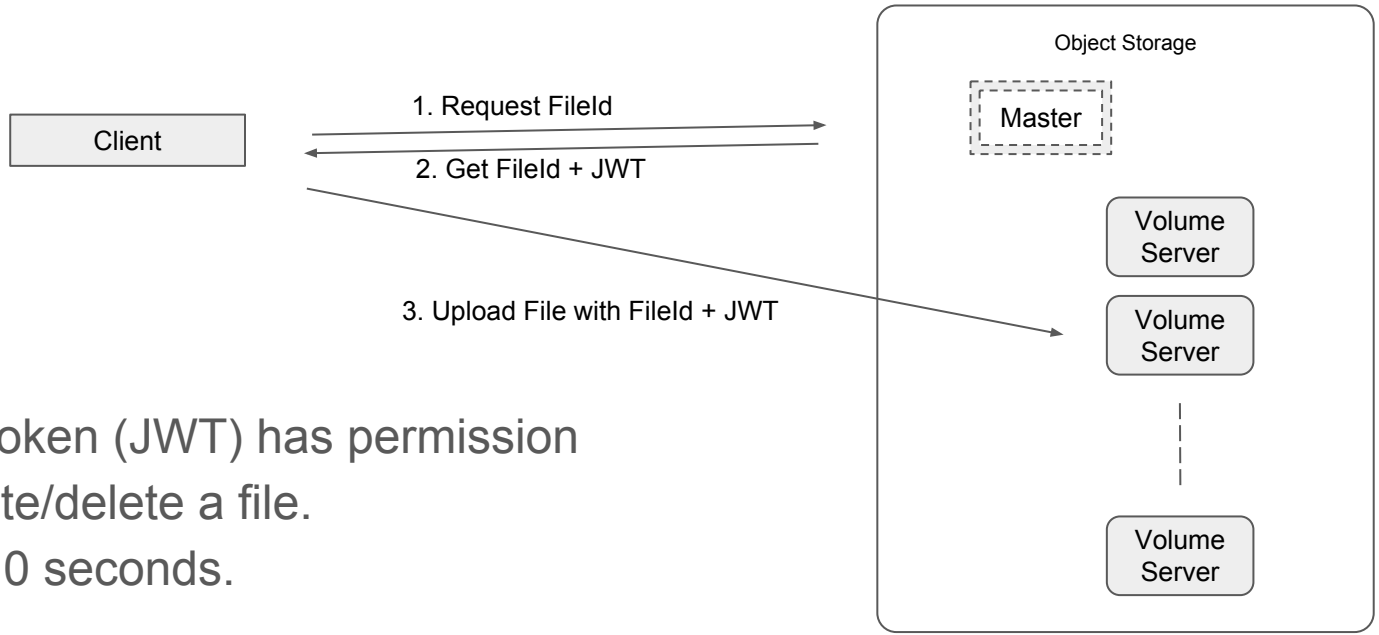
Volume-Aware Clients



Volume-based data placement

- Volumes are organized with different settings:
  - Collection
    - TTL
    - Replication
- Master randomly assigns a write request to one of the writable volumes.
- Strong consistent writes to all replicas.
- If one replica fails heartbeat, the master marks the volume id as read-only.
- Writes should be assigned to other writable volumes.

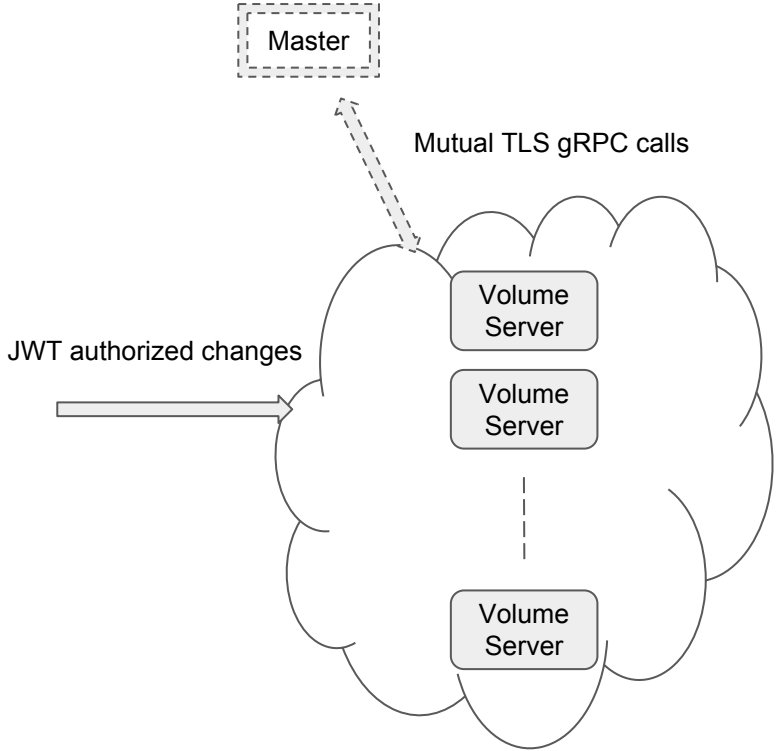
Security: per object access control with JWT



- A Json Web Token (JWT) has permission to create/update/delete a file.
- Expires after 10 seconds.

Secure Volume Server

- Mutual TLS
  - Secure master to volume server admin operations
- JWT
  - Secure object changes

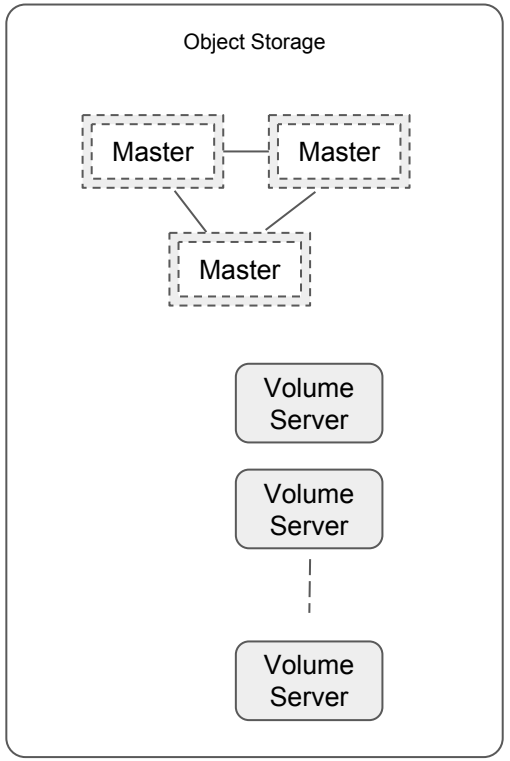


Volume servers can be placed anywhere.

Any server with some free space can be a volume server.

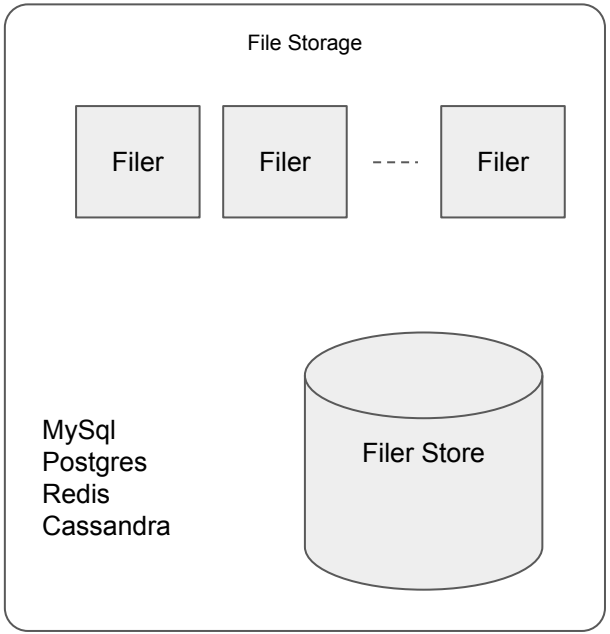
High Availability: Master Server

- Multi-Master cluster
- Leader election with Raft consensus algorithm



High Availability: Filer Server

- Multiple stateless filer servers
- Shared filer store could be any HA storage solution.

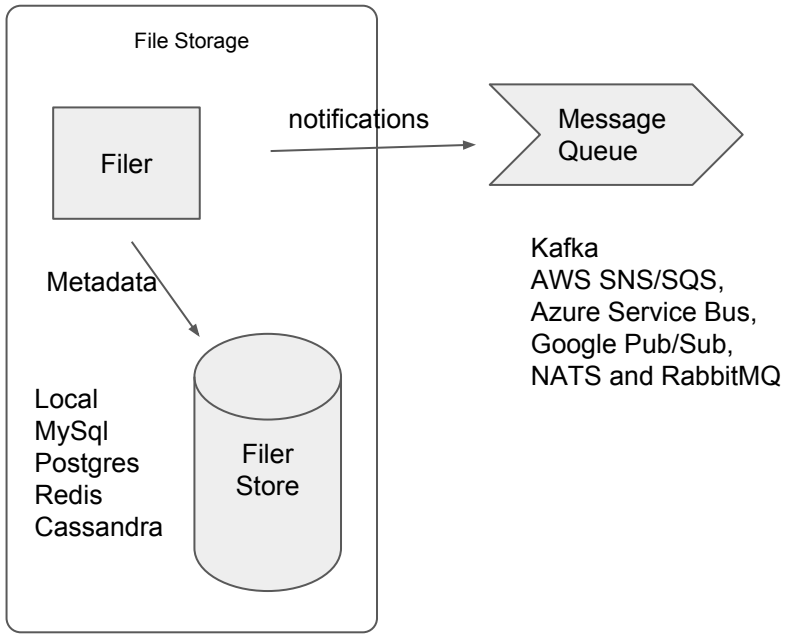


Scalability: Filer

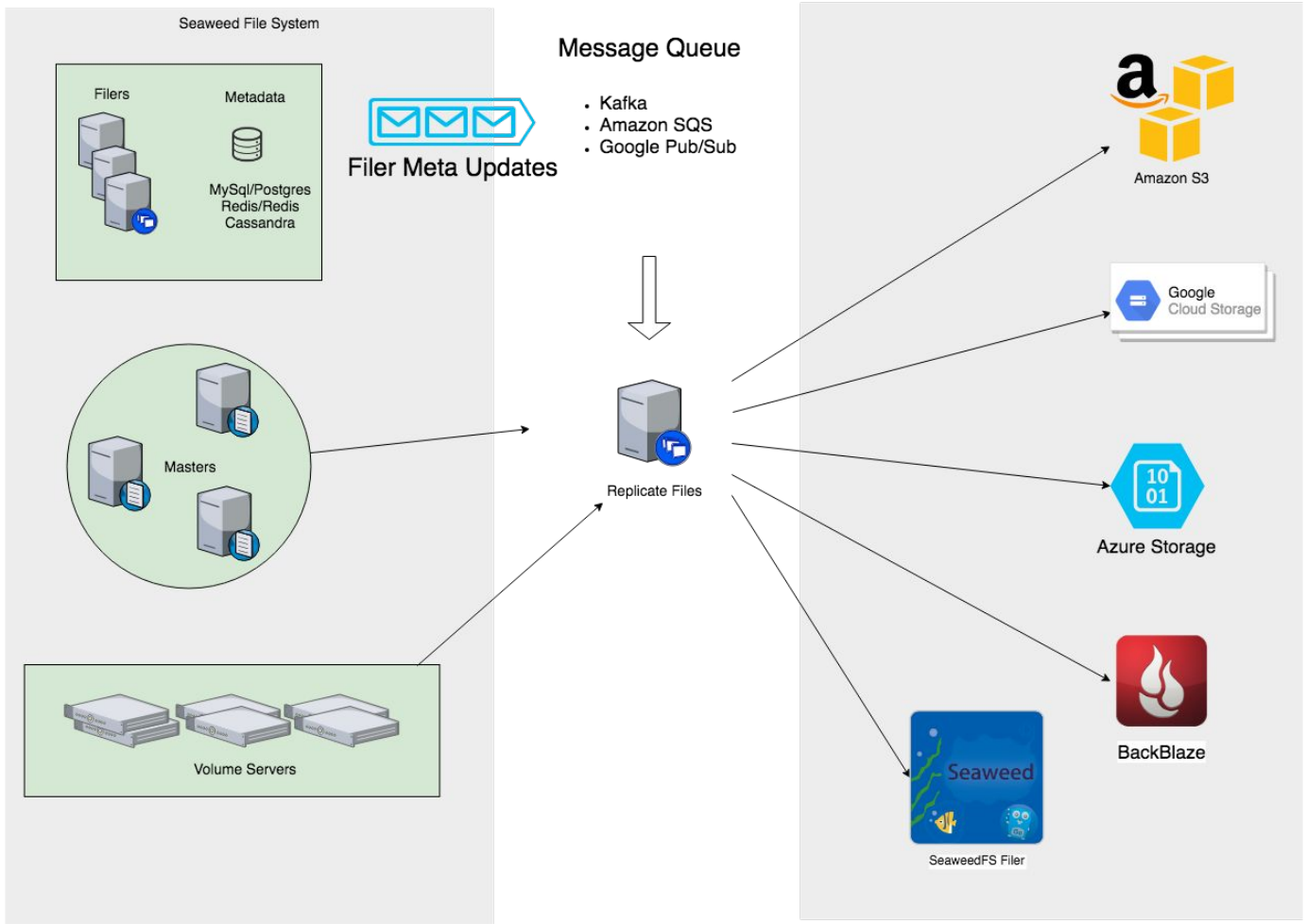
- Direct blob access.
- Filer store can be any proven store, and simple to add new store:
  - Redis
  - MySql/Postgres
  - Cassandra
  - Interface for any key-value store
- Unlimited files under one directory.
- Blob storage supports multiple filers.

File Change Notification

- All filer change notifications can be sent to a message queue.
- Protobuf encoded notification.
- Cross-Region replication is built on top of this.



Seaweed File System Filer Async Replication



Atomicity

Operation	Atomicity	Note
Creating a file	yes	
Deleting a file	yes	
Renaming a file	Yes with mysql/postgres. No with redis/leveldb/cassandra.	Implemented via database transactions.
Renaming a directory	Yes with mysql/postgres. No with redis/leveldb/cassandra.	Implemented via database transactions.
Creating a single directory with mkdir()	yes	
Recursive directory deletion	No	

Comparing to HDFS

	HDFS	SeaweedFS
File Metadata Storage	Single namenode	Multiple stateless filers with proven scalable filer store, redis/cassandra/etc.
Storing small files	Not recommended.	Optimized for small files.
Parallel data access	Yes	Yes
Hadoop Compatible	Yes	Yes. (Atomic rename via database transactions.)

Comparing to CEPH

	CEPH	SeaweedFS
Data Placement	CRUSH maps of the whole cluster, rather complicated, especially when adding storage. Calculated for each object.	Volume level placement, amortized for each object.
Storing small files	Not optimized.	Optimized for small files.
Scaling file system metadata	MDS dynamically partition subtree	Flat and linearly scalable.
Easy to set up	Mixed reviews	Yes

Design Philosophy

- Scale up each layer independently.
- Batch small files
  - Data placement (CEPH file-level, SeaweedFS volume-level)
  - Tracking (HDFS namenode track blocks, SeaweedFS track volume locations)
  - Easy move/delete/replicate operation.

Open APIs

- gRPC APIs for admin operations
- HTTP APIs for uploading and serving blobs
- gRPC for filer metadata operations
- Protocol buffer defined metadata

Future Plan

- Volume Server
  - Async Replica
  - Erasure Coding
  - Tiered Storage
- Integration
  - CSI, docker volume plugin
  - Kerberos
- Tools
  - Auto Balance

Open APIs for possible extensions

- Build a different filer with striping.
- Build a different replication
- Admin tools
- Custom Encryption
- Async Operations
  - Search
  - Secondary index
- Local cache for cloud files
- CDN