SeaweedFS Architecture

Last Edited Time	@August 15, 2021 2:51 AM

Description

SeaweedFS is a distributed storage system for blobs, objects, files, and data warehouse, with predictable low latency with O(1) disk seek, and flexible data placement with multiple tiers and cloud tiers.

SeaweedFS is built into multiple layers.

- **Blob Storage** consists of **master** , **volume** servers, and **cloud tier**.
- **File Storage** consists of the Blob Storage and **filer** servers.
- **Object Storage** consists of the File Storage and **S3** servers.
- **Data warehouse** consists of the File Storage and Hadoop compatible libraries, used by HDFS, Hadoop, Spark, Flink, Presto, HBase, etc.
- **FUSE Mount** consists of the File Storage mounted to the user space on clients, used in common FUSE mount, Kubernetes persistent volume, etc.

SeaweedFS is:

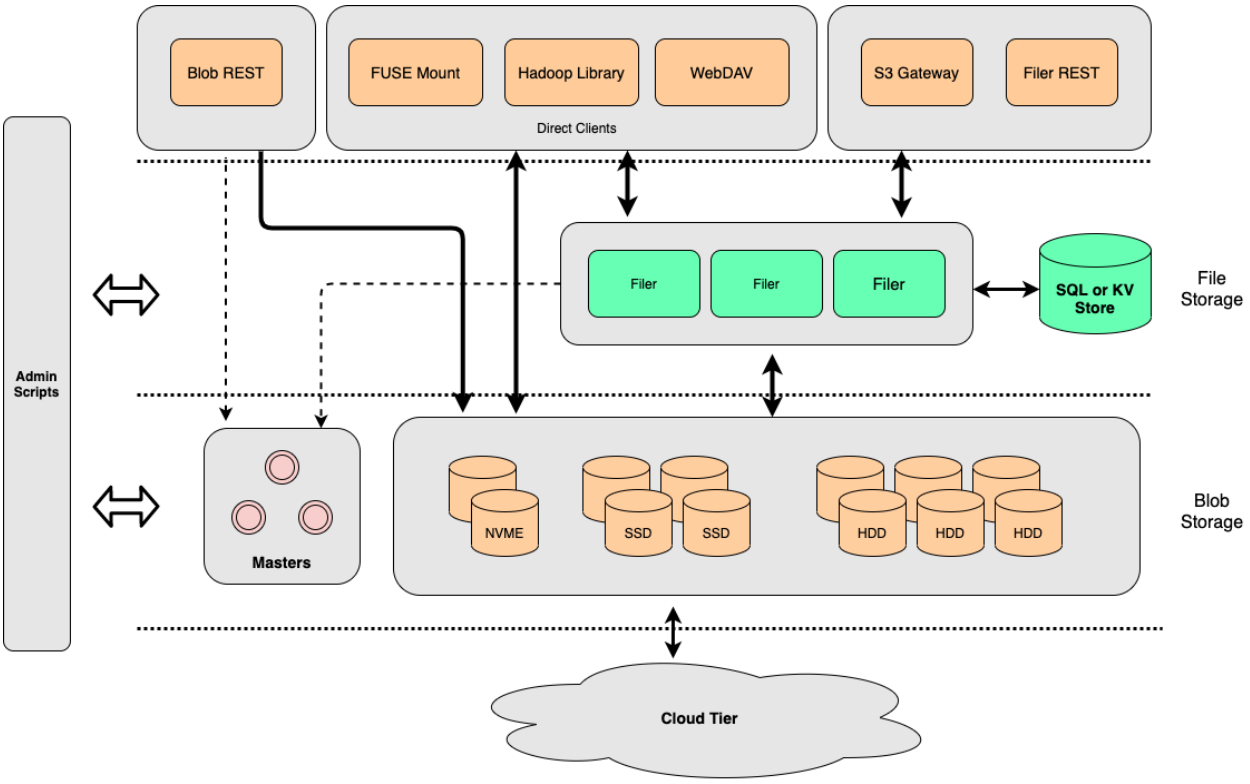
- A fast key to file mapping with O(1) disk seek.
- A customizable tiered storage placing data on demand.
- An elastic storage system that offloads less active data to cloud.
- A scalable file system to replace HDFS.
- A high-performance in-house S3 compatible object store.

The advantages of SeaweedFS includes:

- High data and service availability:
  - No SPOF.
  - Suppors Active-Active asynchronous replication.
  - Supports Erasure Coding.
  - Supports file checksum.
  - Supports rack and data center aware replication
  - Supports metadata backup and replication.
- Optimized for performance:
  - Optimized for lots of small files. Each file overhead is 40 bytes.

    - Always O(1) disk read, even for Erasure-Coded data.
    - Linear scalable. No HDFS name node bottleneck.
    - Clients access data directly on volume servers. Filer servers are accessed for meta data.
    - Place data by requirements to custom disk types, e.g., NVME, SSD, HDD.
- Simple operations
    - Just add volume servers to increase capacity.
    - Filer metadata stored in common systems, e.g., MySql, Postgres, Redis, MongoDB, Cassandra, HBase, ElasticSearch, LevelDB, RocksDB, Etcd, etc.
    - Filer metadata can be migrated or backed up across different stores.
    - Migrating volume data are as simple as copying a few large files.

Diagram



Blob Storage

Volume Server

The volume servers stores a bunch of volumes. Each volume defaults to 30GB. The disk space can be pre-allocated. All the writes are append only. The volumes are compacted periodically. So there are not much disk space wasted.

Master Server

The master servers tracks volume locations on all the volume servers. These information are dynamically collected as soft states and never persisted. These info are provided to clients as a DNS to locate each volume.

Volume

Each volume can contain a lots of files. Each file is indexed by its offset in a separate volume index. So each file read request will just read the in-memory volume index for the offset and size, and then read the volume file with one disk seek.

Volumes can have different replication settings. Each write to one volume will be replicated to its peers with strong consistency. If any replica fails to write, the whole request fails.

Volumes also have different TTL settings. The TTL volumes are not compacted, but just purged as a whole if all the entries in a volume past the TTL.

Volumes can be tagged as different disk types, such as SSD, HDD, or actually any tags, such as NVME, fastHDD, slowHDD. The data are placed according to write request requirement. The admin scripts can move the data to other tiers.

Collection

One collection is just a bunch of volumes with a common collection name. A collection is easier to manage. For example, each S3 bucket has its own collection of volumes. When one bucket can be dropped, we just need to remove all the volumes in the collection to purge the file data.

One collection can have multiple volume types, e.g., different replication, different TTL, different disk types.

File Id

A file id has 3 parts: <volume id, file key, file cookie>.

Read and Write process

Each write request will ask master for a file id, which contains a writable volume matching the required replication, TTL, and diskType, and then send the file content to the volume server directly.

The writing client should store the file id in some database.

Each read request will also ask master for the volume server location for the file id, and then read from the volume server directly.

Replication

SeaweedFS implements strong consistency for writes.

- One client will request the master to assign a volume id with a replication factor, e.g., "010".
- With the assigned volume id, the client will send the write request to one of the volume servers holding the volume.
- The volume server will append the file content to the volume, and also replicate the content to other replicas of this volume. If any one of the writes failed, the volume server will report error to the client.
- The client will complete on success writes, or restart from beginning to write to another volume id.

The assumption is there should be enough volumes in the system. If some of them failed due to transient changes, other volumes can be assigned.

For read requests, the client just should pick one replica, and retry on the next replica if fails.

The replication is at volume level. Customizable admin scripts will fix missing volume replicas, balance volume servers, etc.

The strong consistent writes are for hot data, which needs fewer number of network hops. Erasure coding can be applied to cold data, which adds redundancy and still keeps O(1) disk seek for reads.

Advantages

These read and write processes used the volume id to locate the volume. This extra level of indirection provides flexibility to move the volume around, without any strict data placement algorithm requirements.

In addition, the O(1) disk seek minimized the cost to read.

These two features enabled SeaweedFS to flexibly place volumes on any tier, even to cloud tier, yet still with minimum impact to performance.

Common Use cases

- Save images in the blob store and save file id in a database.
- Build more complicated storage systems on top of it.

File Storage

Filer Store

In addition to file content, a file system also needs directories and files. These metadata are stored in a separate filer store. SeaweedFS uses existing data stores instead of re-inventing them.

Most SQL and non-SQL stores are supported, including

- SQL and their variations
  - Postgres, YugabyteDB, CockroachDB
  - MySQL, MariaDB MemSQL, TiDB,
- Non-SQL
  - Redis, Tendis, Ledis
  - Cassandra, ScyllaDB
  - HBase
  - MongoDB
  - ElasticSearch
  - Etcd

- Embedded
  - LevelDB
  - RocksDB

Filer

Filer is designed as stateless. For read and writes, it just access the filer store for metadata, and access the volume servers to read or write.

Filer also connects to master servers for up-to-date volume locations, and requests for file id during writes.

For embedded filer stores, the filers also need to communicate with each other to share the metadata changes.

For large files, filer will chunk the file and saved the chunks to volume servers. If the number of chunks is large, the list of chunk ids will also be saved as a meta chunk. So there are no issue with large or super large files.

File content can be encrypted with the key stored as metadata. So the volume data can be safely stored anywhere, on any other servers or in the cloud.

Metadata Change Event Subscription

All metadata changes are saved as change log files, stored in the SeaweedFS itself.

Besides that, the metadata change events can be subscribed via gRPC calls. This asynchronous metadata update powers advanced features, such as flexible replication, streaming backup, fast local metadata reads in fuse mount, metadata hot replica, etc.

Being in gRPC, other languages can also observe metadata events for any directory tree, and build large scale event-driven applications.

Lots Of Small Files

Lots of small files has been a big challenge for HDFS system and many other big data systems. SeaweedFS addresses it from 2 aspects:

- Compact file storage:
  - Files are appended consecutively in big volumes: Minimizing native file system's inode operations. One file disk overhead is less than 40 bytes, while

one *xfs_inode_t* structure in Linux is 536 bytes.

    - Operations are applied at the volume level, including Erasure Coding, replication, balancing, etc.
- Scalable metadata management: With the external metadata store, SeaweedFS can scale metadata management in multiple approaches.
    - Add path-specific metadata stores.
    - Add more metadata store servers if supported.
    - Add more filers.

Filer as a key-large-value store

Conventionally, file systems are for storing files. The high cost of inode prohibits from storing too many files in a file system. However, with SeaweedFS, each file is stored as one data store entry, pointing to file content in volume servers. With the optimization for small files, we can use SeaweedFS as a path to large value store.

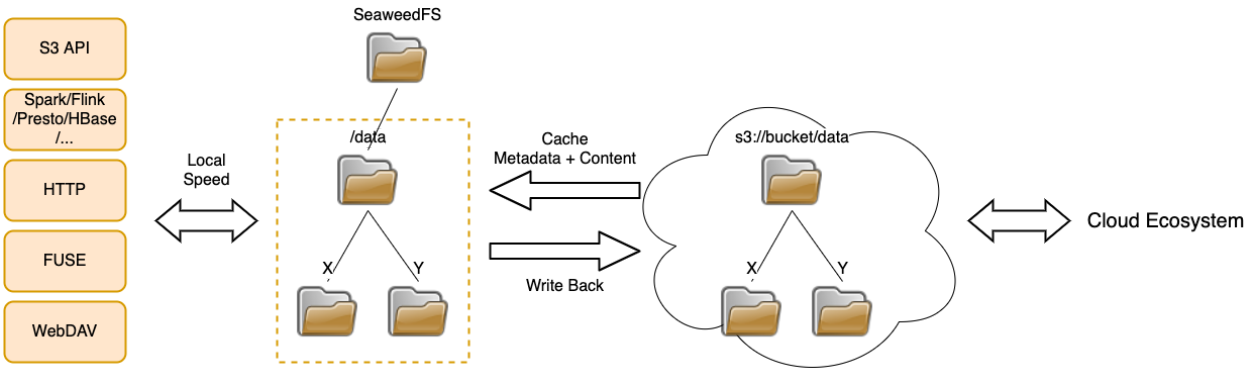
Super Large Directories

SeaweedFS supports super large directories. There are no limits in terms of the number of children under that directory.

The caveat is that the directory listing is not supported(yet).

Remote Storage Cache

To speed up data access to data already on cloud, SeaweedFS can mount any cloud folder to local.



During the mounting process, the cloud metadata is pulled down and saved as normal metadata.

To manage metadata, there is an admin command to refresh metadata changes from the cloud storage.

To manage file content, there are optional admin commands to cache or uncache file content for any folder, any file name pattern, any file size range, and any file age range.

If the file content is not cached yet, the filer will let volume servers read the the cloud data chunks in parallel.

The cached data are treated as normal metadata and data. They are read or written at local network speed.

Any local updates are asynchronously written back to the cloud storage, by a separate `weed filer.remote.sync` process.

This cloud data caching will not change data format. So the data work well with existing cloud ecosystems.

FUSE Mount

FUSE Mount is mostly POSIX compatible.

The mount keeps a local metadata cache. The cache asynchronously subscribes to filer metadata changes. So any metadata operations such as directory listing, traversing, are very fast with just local reads.

The metadata updates are written to filer store via gRPC calls first, and then saved to local metadata cache. So metadata is always consistent.

Object Storage

Object store implements S3 compatible APIs. Most common APIs are supported. The access key and secret key can be dynamically configured.

Each bucket is created with its own collection of volumes. Optionally, bucket metadata can be stored in dedicated SQL tables or databases for supporting metadata stores. So dropping a bucket can be instant.

With path-specific configuration, each bucket can also have its own storage requirements, such as different replication, TTL, disk type, concurrency.

Hadoop Compatible File System

SeaweedFS provides java libraries compatible with Hadoop systems. Most systems running on Hadoop, e.g., Spark, Flink, Presto, HBase, etc, can be migrated to SeaweedFS, by adding one jar file to the client's classpath.

APIs

Blob and File storage layers can be accessed directly via basic HTTP requests. There are many existing libraries for various languages.

For Java, there is a Java client for file storage also, which is refactored out of the SeaweedFS Hadoop library.

Master, Volume, and Filer servers all have gRPC open APIs.

Replication and Backup

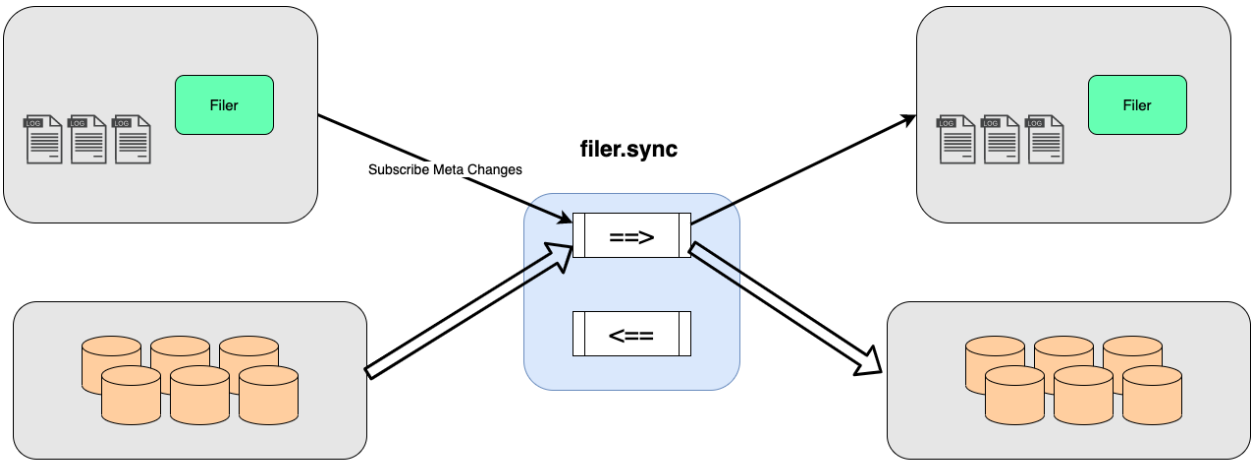
SeaweedFS provides different level of replication or backup features.

- Cluster active-active XDC replication.
- Cluster backup to other machines, or to cloud storage.
- Volume data replication and erasure-coding.
- Filer metadata store backup.

Cluster Active-Active Replication

SeaweedFS clusters can be replicated across data centers in both directions.

- All the metadata changes are persisted as log files, and can be subscribed.
- A "filer.sync" process subscribes the metadata changes, and replicated the metadata and file content over to the target cluster.
- Replication progress is check-pointed periodically and can be resumed if interrupted.
- Replication can be setup in active-active mode or active-passive mode, for any specific folder.
- The replications can be bi-directional. All metadata changes are idempotent and all events have signatures to prevent loops.

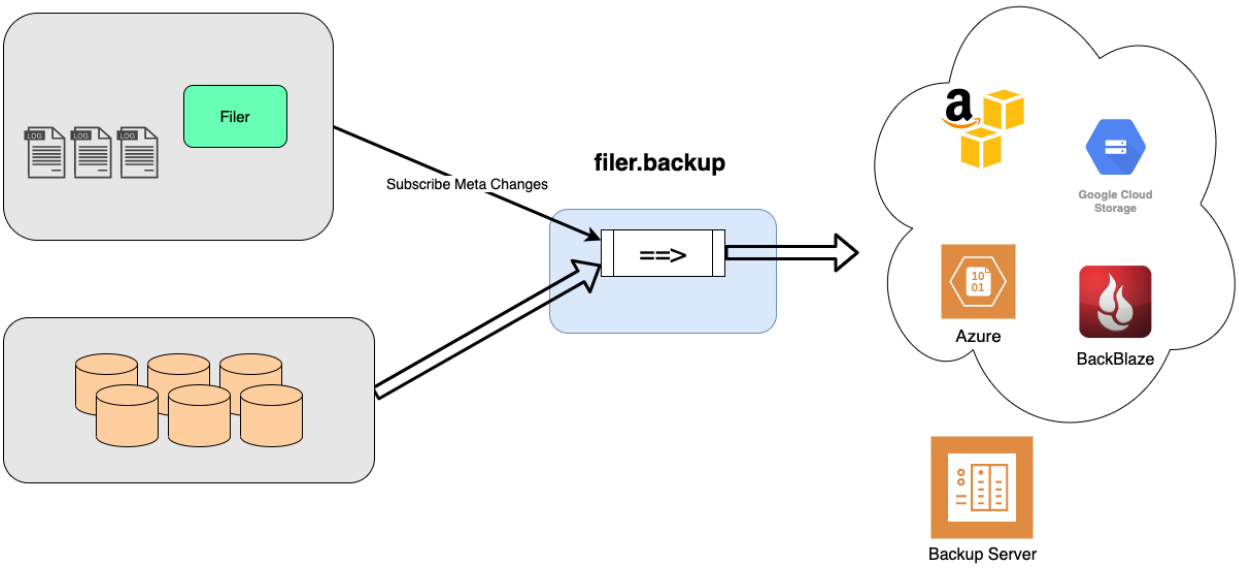


Because of the metadata signatures, multiple clusters can be connected together with "filer.sync" processes, without the replication loops that may melt down the system. More complex topologies of replication can be formed, e.g, a ring configuration, a star configuration, or a Y-split configuration.

Cluster Backup

The "filer.backup" is similar to "filer.sync", but it can backup files to other servers or to cloud storage.

An optional incremental mode can save new or modified content for each date, while skipping deletions.



Tiered Storage and Cloud Tier

The volume index files are small but need quick access. They can optionally be stored in fast disks for optimal performance.

The admin scripts can be run regularly to find and move less active volumes to cheaper and slower storage tiers. The volume index files will stay on local fast disks. Since all file content reads are O(1) disk seek, there are no multiple round trips to read one file and have minimum performance impact.

If local disk spaces run out, the data can be offloaded to cloud storage. Cloud storage usually does not charge for uploading traffic. So it is ideal for backup. However, in addition to storage cost, cloud service usually also changes API fees. With O(1) read operation, SeaweedFS can keep the API cost to minimum.

Administration

SeaweedFS provides "weed shell". The administration scripts can be run interactively in the shell, or let the shell process the script file.

- S3 Maintenance

    - configure credentials and permissions for S3 buckets
    - list, create or delete buckets
- Filer Maintenance
    - configure path-specific storage options
    - list and cat files
    - export and import metadata
- Volume Maintenance
    - fix volume replication
    - balance volumes
    - move volumes to different tiers or cloud
    - erasure encode or decode volumes
    - mount or unmount volumes
    - vacuum volumes
    - evacuate a volume server, etc.

Security

All gRPC communications between masters, volume servers, and filers can be protected via gRPC TLS connections.

HTTP reads and writes can be protected by JWT.

Kubernetes

SeaweedFS provides two Kubernetes integrations:

- Kubernetes Operator provides easy deployment of SeaweedFS clusters.
- Kubernetes CSI provides ReadWriteMany persistent volumes.