GSoC 2024 Apache OpenDAL OVFS Project Proposal

1 Basic Information

- **Name:** Runjie Yu
- **Email:** zjregee@gmail.com
- **Github:** https://github.com/zjregee
- **Location:** Wuhan, China (GMT+8:00)
- **University:** Huazhong University of Science and Technology

2 Project Information

- **Project Name:** OVFS, OpenDAL File System via Virtio
- **Related Issues:** https://github.com/apache/opendal/issues/4133
- **Project Mentors:** Xuanwo (xuanwo@apache.org), Manjusaka (manjusaka@apache.org)

- **Project Community:** Apache OpenDAL
- **Project Difficulty:** Major
- **Project Size:** Large, ~350 hours

3 About Me

I am Runjie Yu, a first-year master's student majoring in Computer Science and Technology at Huazhong University of Science and Technology in China. I also earned my undergraduate degree from the same university. I have a strong passion for programming and have experience in various programming languages, including Rust, Go, C++, Swift, and more. My primary interest lies in studying system development, particularly in the areas of file systems and databases, and I am eager to achieve meaningful research results. I have completed internships at Tencent and ByteDance. Coding is not merely a skill for me, I preceive it as a long-term career.

I believe OpenDAL is an outstanding Apache project. It provides a unified, efficient, and cost-effective data access interface for numerous storage services. This project can seamlessly integrate into varius systems as a storage layer, holding significant potential and value in the prevailing era of cloud-native technologies. I am determined to contribute my best efforts to its development through continuous work.

4 Project Abstract

Virtio is an open standard designed to enhance I/O performance between virtual machines (VMs) and host systems in virtualized environments. VirtioFS is an extension of the Virtio standard specifically crafted for file system sharing between VMs and the host. This is particularly beneficial in scenarios where seamless access to shared files and data between VMs and the host is essential. VirtioFS has been widely adopted in virtualization technologies such as QEMU and Kata Container.

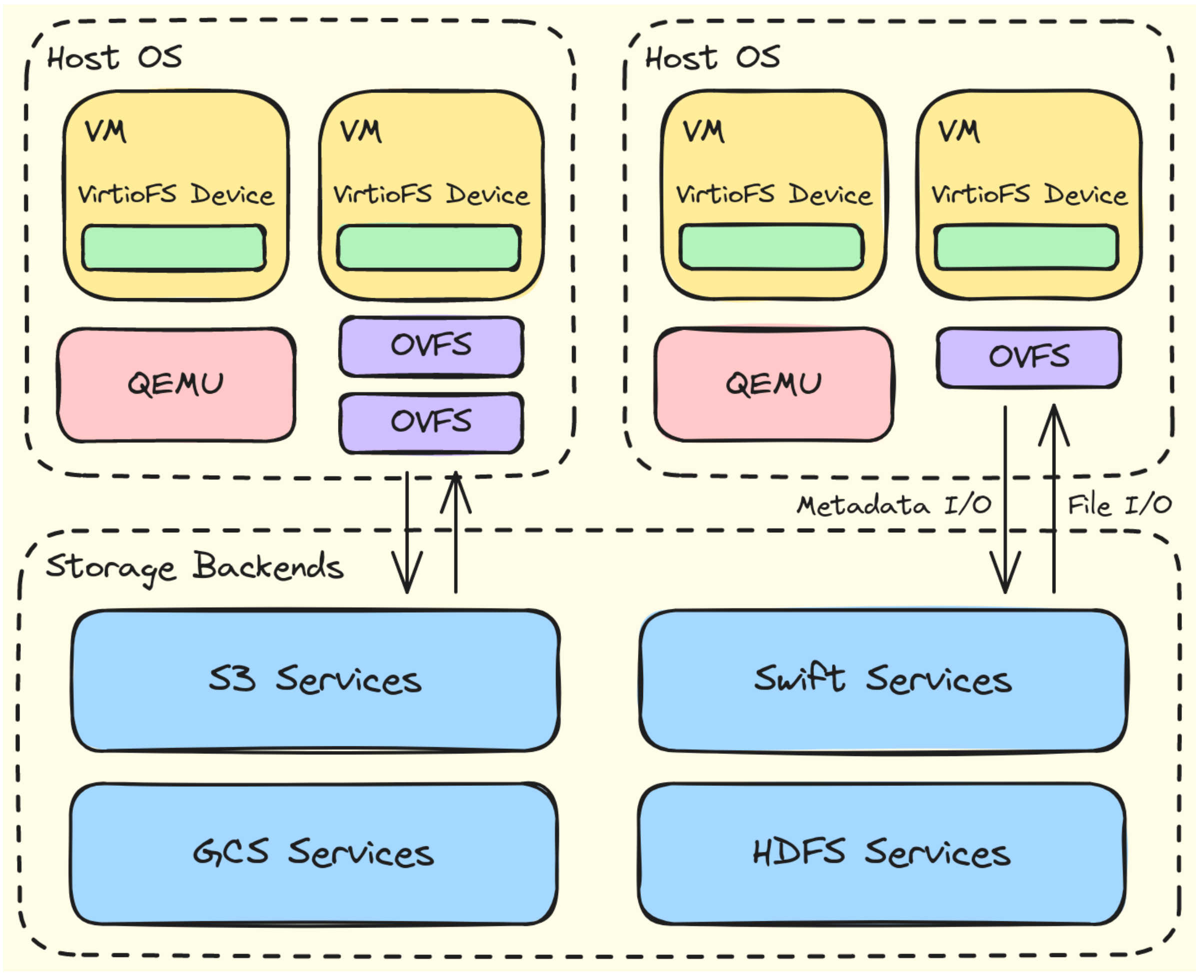
Apache OpenDAL is a data access layer that allows users to easily and efficiently retrieve data from various storage services in a unified manner. In this project, our goal is to reference virtiofsd (a standard vhost-user backend, a pure Rust implementation of VirtioFS based on the local file system) and implement VirtioFS based on OpenDAL.

Through this project, VMs can access numerous data services through the file system interface with the assistance of the OpenDAL service daemon deployed on the host, all without their awareness. It ensures the efficiency of file system reading and writing in VMs through VirtioFS support. This storage-system-as-a-service approach conceals the details of the distributed storage system from VMs. This ensures the security of storage services, as VMs do not need to be aware of the information, configuration and permission credentials of the accessed storage service. Additionally, it enables the utilization of a new backend storage system without reconfiguring all VMs.

5 Project Detailed Description

This chapter serves as an introduction to the overall structure of the project, outlining the design ideas and principles of critical components. It covers the OVFS architecture, interaction principles, design philosophy, file system operations based on various storage backend, cache pool design, configuration support, potential usage scenarios of OVFS, and the expected file system interface support.

5.1 The Architecture of OVFS



The picture above is the OVFS architecture diagram. OVFS is a file system implementation based on the VirtioFS protocol and OpenDAL. It serves as a bridge for semantic access to file system interfaces between VMs and external storage systems. Leveraging the multiple service access capabilities and unified abstraction provided by OpenDAL, OVFS can conveniently mount shared directories in VMs on various existing distributed storage services.

The complete OVFS architecture consists of three crucial components:

- VMs FUSE client that supports the VirtioFS protocol and implements the VirtioFS Virtio device specification. An appropriately configured Linux 5.4 or later can be used for OVFS. The VirtioFS protocol is built on FUSE and utilizes the VirtioFS Virtio device to transmit FUSE messages. In contrast to traditional FUSE, where the file system daemon runs in the guest user space, the VirtioFS protocol supports forwarding file system requests from the guest to the host, enabling related processes on the host to function as the guest's local file system.
- A hypervisor that implements the VirtioFS Virtio device specification, such as QEMU. The hypervisor needs to adhere to the VirtioFS Virtio device specification, supporting devices used during the operation of VMs, managing the file system operations of the VMs, and delegating these operations to a specific vhost-user device backend implementation.
- A vhost-user backend implementation, namely OVFS. This is a crucial aspect that requires particular attention in this project. This backend is a file system daemon running on the host side, responsible for handling all file system operations from VMs to access the shared directory. virtiofsd offers a practical example of a vhost-user backend implementation, based on pure Rust, forwarding VMs' file system requests to the local file system on the host side.

5.2 How OVFS Interacts With VMs and Hypervisor

The Virtio specification defines device emulation and communication between VMs and the hypervisor. Among these, the virtio queue is a core component of the communication mechanism in the Virtio specification and a key mechanism for achieving efficient communication between VMs and the hypervisor. The virtio queue is essentially a shared memory area called vring between VMs and the hypervisor, through which the guest sends and receives data to the host.

Simultaneously, the Virtio specification provides various forms of Virtio device models and data interaction support. The vhost-user backend implemented by OVFS achieves information transmission through the vhost-user protocol. The vhost-user protocol enables the sharing of virtio queues through communication over Unix domain sockets. Interaction with VMs and the hypervisor is accomplished by listening on the corresponding sockets provided by the hypervisor.

In terms of specific implementation, the vm-memory crate, virtio-queue crate and vhost-user-backend crate play crucial roles in managing the interaction between OVFS, VMs, and the hypervisor.

The vm-memory crate provides encapsulation of VMs memory and achieves decoupling of memory usage. Through the vm-memory crate, OVFS can access relevant memory without knowing the implementation details of the VMs memory. Two formats of virtio queues are defined in the Virtio specification: split virtio queue and packed virtio queue. The virtio-queue crate provides support for the split virtio queue. Through the DescriptorChain package provided by the virtio-queue crate, OVFS can parse the corresponding virtio queue structure from the original vring data. The vhost-user-backend crate provides a way to start and stop the file system demon, as well as encapsulation of vring access. OVFS implements the vhost-user backend service based on the framework provided by the vhost-user-backend crate and implements the event loop for the file system process to handle requests through this crate.

5.3 OVFS Design Philosophy

In this section, we will present the design philosophy of the OVFS project. The concepts introduced here will permeate throughout the entire design and implementation of OVFS, fully manifesting in other sections of the proposal.

Stateless Services

The mission of OVFS is to provide efficient and flexible data access methods for VMs using Virtio and VirtioFS technologies. Through a stateless services design, OVFS can easily facilitate large-scale deployment, expansion, restarts, and error recovery in a cluster environment running multiple VMs. This seamless integration into existing distributed cluster environments means that users do not need to perceive or maintain additional stateful services because of OVFS.

To achieve stateless services, OVFS refrains from persisting any metadata information. Instead, it maintains and synchronizes all state information of the OVFS file system during operation through the backend storage system. There are two implications here: OVFS doesn't need to retain additional operational status during runtime; it doesn't require the maintenance of additional file system metadata when retrieving data from the backend storage system. Consequently, OVFS doesn't necessitate exclusive access to the storage system. It permits any other application to read and write data to the storage system when it serves as the storage backend for OVFS. Furthermore, OVFS ensures that the usage semantics of data in the storage system remain unchanged. All data in the storage system is visible and interpretable to other external applications.

Under this design, OVFS alleviates concerns regarding synchronization overhead and potential consistency issues stemming from data alterations in the storage system due to external operations, thereby reducing the threshold and risks associated with OVFS usage.

Storage System As A Service

We aspire for OVFS to serve as a fundamental storage layer within a VM cluster. With OVFS's assistance, VMs can flexibly and conveniently execute data read and write operations through existing distributed storage system clusters. OVFS enables the creation of different mount points for various storage systems. This service design model is extremely scalable and supports VMs to access multiple existing storage systems. By accessing different mount points, VMs can seamlessly access various storage services.

This design pattern allows users to customize the data access pipeline of VMs in distributed clusters according to their needs and standardizes the data reading, writing, and synchronization processes of VMs. In case of a network or internal error in a mounted storage system, it will not disrupt the normal operation of other storage systems under different mount points.

User-Friendly Interface

OVFS must offer users a user-friendly operating interface. This entails ensuring that OVFS is easy to configure, intuitive, and controllable in terms of behavior. OVFS accomplishes this through the following aspects:

- It's essential to offer configurations for different storage systems that align with OpenDAL. For users familiar with OpenDAL, there's no additional learning curve.
- OVFS is deployed using a formatted configuration file format. The operation and maintenance of OVFS only require a TOML file with clear content.
- Offer clear documentation, including usage and deployment instructions, along with relevant scenario descriptions.

5.4 File System Operations Based On Various Storage Backend

OVFS implements a file system model based on OpenDAL. A file system model that provides POSIX semantics should include access to file data and metadata, maintenance of directory trees (hierarchical relationships between files), and additional POSIX interfaces.

Lazy Metadata Fetch In OVFS

OpenDAL natively supports various storage systems, including object storage, file storage, key-value storage, and more. However, not all storage systems directly offer an abstraction of file systems. Take AWS S3 as an example, which provides object storage services. It abstracts the concepts of buckets and objects, enabling users to create multiple buckets and multiple objects within each bucket. Representing this classic two-level relationship in object storage directly within the nested structure of a file system directory tree poses a challenge.

To enable OVFS to support various storage systems as file data storage backends, OVFS will offer different assumptions for constructing directory tree semantics for different types of storage systems to achieve file system semantics. This design approach allows OVFS to lazily obtain metadata information without the need to store and maintain additional metadata. Additional metadata not only leads to synchronization and consistency issues that are challenging to handle but also complicated OVFS's implementation of stateless services. Stateful services are difficult to maintain and expand, and they are not suitable for potential virtualization scenarios of OVFS.

File System Operations Based On Object Storage Backend

The working principle of OVFS based on the object storage backend is to implement the file and directory in the file system through a single bucket in the object storage. When using OVFS, users need to specify the bucket used to store data. A comprehensive directory tree architecture is realized by treating the object name in the bucket as a full path in the file system and treating the slash character "/" as a directory separator. File system operations in the VMs can interact with the object storage system through similar escape operations to achieve file system interface based data reading and writing.

The following table lists the mapping of some file system operations based on the object storage system. Here we refer to the mapping method of FUSE operations to object storage system operations in Google Cloud Storage FUSE.

File System Operations	Object Storage Backend Operations

read all entries under the directory with the full path "/xxx/yyy"	Objects.get("/xxx/yyy/")
	Objects.list("/xxx/yyy/")

create a directory with the full path "/xxx/yyy"	Objects.get("/xxx/yyy")
	Objects.get("/xxx/yyy/")
	Objects.insert("/xxx/yyy/")

remove a directory with the full path "/xxx/yyy" and delete all entries under the directory	Objects.get("/xxx/yyy/")
	Objects.list("/xxx/yyy/")
	Objects.delete("/xxx/yyy/local.txt")
	Objects.delete("/xxx/yyy/")

create a file named "zzz" in a directory with the full path "/xxx/yyy"	Objects.get("/xxx/yyy/")
	Objects.get("/xxx/yyy/zzz")
	Objects.get("/xxx/yyy/zzz/")
	Objects.insert("/xxx/yyy/zzz")

remove a file named "zzz" in a directory with the full path "/xxx/yyy"	Objects.get("/xxx/yyy/")
	Objects.get("/xxx/yyy/zzz")
	Objects.delete("/xxx/yyy/zzz")

File System Operations Based On File Storage Backend

Unlike distributed object storage systems, distributed file systems already offer operational support for file system semantics. Therefore, OVFS based on a distributed file system doesn't require additional processing of file system requests and can achieve file system semantics simply by forwarding requests.

Limitations Under OVFS Metadata Management

While OVFS strives to implement a unified file system access interface for various storage system backends, users still need to be aware of its limitations and potential differences. OVFS supports a range of file system interfaces, but this doesn't imply POSIX standard compliance. OVFS cannot support some file system calls specified in the POSIX standard.

5.5 Multi Granular Object Size Cache Pool

In order to improve data read and write performance and avoid the significant overhead caused by repeated transmission of hot data between the storage system and the host, OVFS needs to build a data cache in the memory on the host side.

Cache Pool Based On Multi Linked List

OVFS will create a memory pool to cache file data during the file system read and write process. This huge memory pool is divided into object sizes of different granularities (such as 4 kb, 16 kb, 64 kb, etc.) to adapt to different sizes of data file data blocks.

Unused cache blocks of the same size in the memory pool are organized through a linked list. When a cache block needs to be allocated, the unused cache block can be obtained directly from the head of the linked list. When a cache block that is no longer used needs to be recycled, the cache block is added to the tail of the linked list. By using linked lists, no only can the algorithmic complexity of allocation and recycling be O(1), but furthermore, lock-free concurrency can be achieved by using CAS operations.

Write Back Strategy

OVFS manages the data reading and writing process through the write back strategy. Specifically, when writing data, the data is first written to the cache, and the dirty data will be gradually synchronized to the backend storage system in an asynchronous manner. When reading the file data, the data will be requested from the backend storage system after a cache miss or expiration , and the new data will be updated to the cache, and its expiration time will be set.

OVFS will update the dirty data in the cache to the storage system in these cases:

- When VMs calles fysnc, fdatasync, or used related flags during data writing.
- The cache pool is full, and dirty data needs to be written to make space in the cache. This is also known as cache eviction, and the eviction order can be maintained using the LRU algorithm.
- Cleaned by threads that regularly clean dirty data or expired data.

DAX Window Support (Experimental)

The VirtioFS protocol extends the DAX window experimental features based on the FUSE protocol. This feature allows memory mapping of file contents to be supported in virtualization scenarios. The mapping is set up by issuing a FUSE request to OVFS, which then communicates with QEMU to establish the VMs memory map. VMs can delete mapping in a similar manner. The size of the DAX window can be configured based on available VM address space and memory mapping requirements.

By using the mmap and memfd mechanisms, OVFS can use the data in the cache to create an anonymous memory mapping area and share this memory mapping with VMs to implement the DAX Window. The best performance is achieved when the file contents are fully mapped, eliminating the need for file I/O communication with OVFS. It is possible to use a small DAX window, but this incurs more memory map setup/removal overhead.

5.6 Flexible Configuration Support

Running QEMU With OVFS

As described in the architecture, deploying OVFS involves three parts: a guest kernel with VirtioFS support, QEMU with VirtioFS support, and the VirtioFS daemon (OVFS). Here is an example of running QEMU with OVFS:

```
host# ovfs --config-file=./config.toml

host# qemu-system \
        -blockdev file,node-name=hdd,filename=<image file> \
        -device virtio-blk,drive=hdd \
        -chardev socket,id=char0,path=/tmp/vfsd.sock \
        -device vhost-user-fs-pci,queue-size=1024,chardev=char0,tag=<fs tag> \
        -object memory-backend-memfd,id=mem,size=4G,share=on \
        -numa node,memdev=mem \
        -accel kvm -m 4G

guest# mount -t virtiofs <fs tag> <mount point>
```

The configurations above will generate two devices for the VMs in QEMU. The block device named hdd serves as the backend for the virtio-blk device within the VMs. It functions to store the VMs' disk image files and acts as the primary device within the VMs. Another character device named char0 is implemented as the backend for the vhost-user-fs-pci device using the VirtioFS protocol in the VMs. This character device is of socket type and is connected to the file system daemon in OVFS using the socket path to forward file system messages and requests to OVFS.

It is worth noting that the configuration method largely refers to the configuration in virtiofsd. For the purpose and planning of the project, we ignored many VMs configurations related file system access permissions or boundary handling methods.

Enable Different Distributed Storage Systems

In order for OVFS to utilize the extensive service support provided by OpenDAL, the corresponding service configuration file needs to be provided when running OVFS. The parameters in the configuration file are used to support access to the storage system, including data root address and permission authentication. Below is an example of a configuration file, using a TOML format similar to oli (a command line tool based on OpenDAL):

```
[socket_settings]
socket_path = "/tmp/vfsd.sock"

[cache_settings]
enabled_cache = true
enabled_cache_write_back = false
enabled_cache_expiration = true
cache_expiration_time = "60s"

[storage_settings]
type = "s3"
bucket = "<bucket>"
region = "us-east-1"
endpoint = "https://s3.amazonaws.com"
```

```
access_key_id = "<access_key_id>"
secret_access_key = "<secret_access_key>"
```

OVFS can achieve hot reloading by monitoring changes in the configuration file. This approach allows OVFS to avoid restarting the entire service when modifying certain storage system access configurations and mounting conditions.

5.7 Potential Usage Scenarios

In this section, we list some potential OVFS usage scenarios and application areas through the detailed description of the OVFS project in the proposal. It's worth mentioning that as the project progresses, more application scenarios and areas of advantage may expand, leading to a deeper understanding of the positioning of the OVFS project.

- Unified data management basic software within distributed clusters.
- The OVFS project could prove highly beneficial for large-scale data analysis applications and machine learning training projects. It offers a means for applications within VM clusters to read and write data, models, checkpoints, and logs through common file system interfaces across various distributed storage systems.

5.8 Expected File System Interface Support

Finally, the table below lists the expected file system interface support to be provided by OVFS, along with the corresponding types of distributed storage systems used by OpenDAL.

Command	Object Storage	File Storage	Key-Value Storage
stat	Support	Support	Not Support
touch/rm	Support	Support	Not Support
mkdir/rmdir	Support	Support	Not Support
read/write	Support	Support	Not Support
truncate	Support	Support	Not Support
readdir	Support	Support	Not Support
rename	Support	Support	Not Support
flush/fsync	Support	Support	Not Support
ln	Not Support	Not Support	Not Support
getxattr/setxattr	Not Support	Not Support	Not Support
chmod/chown	Not Support	Not Support	Not Support
access	Not Support	Not Support	Not Support

It is worth mentioning that although storage systems are simply divided into three types here, the file system interface support provided by OVFS needs to be discussed in detail based on the specific service type. In order to avoid introducing too much complexity here, we will use S3 and local file systems as references for object storage systems and file system storage systems respectively, and aim to implement the planning in the above table based on these two typical storage systems.

Since the data volume of an individual file may be substantial, contradicting the design of key-value storage, we do not intend to include support for key-value Storage in this project. In addition, the complex permission system control of Linux is not within the scope of this project. Users can restrict file system access behavior based on the configuration of storage system access permissions in the OVFS configuration file.

6 Deliverables

This chapter describes the items that the OVFS project needs to deliver during the implementation cycle of GSoC 2024.

1. A code repository that implements the functions described in the project details. The services implemented by OVFS in the code repository need to meet the following requirements:
    - VirtioFS implementation, well integrated with VMs and QEMU, able to correctly handle VMs read and write requests to the file system.
    - Supports the use of distributed object storage systems and distributed file systems as storage backends, and provides complete and correct support for at least one specific storage service type for each storage system type. S3 can be used as the target for object storage systems, and local file systems can be used as the target for file systems.
    - Supports related configurations of various storage systems. Users can configure storage system access and use according to actual needs. When an error occurs, users can use the configuration file to restart services.
2. Form an OVFS related test suite. Testing about the project should consist of two parts:
    - Unit testing in code components. Unit testing is the guarantee that the code and related functions are implemented correctly. This test implementation accompanies the entire code implementation process.
    - CI testing based on github actions. The OpenDAL project integrates a large number of CI tests to ensure the correct behavior of OpenDAL under various storage backends. OVFS needs to use good CI testing to check potential errors during code submission.
3. A performance test report of OVFS. The report needs to perform basic metadata operations, data reading and writing performance tests on the VMs mounted with OVFS, and summarize the performance of OVFS through the test results. Reports can be based on file system performance testing tools such as fio, sysbench and mdtest, and compared with virtiofsd when necessary.
4. Documentation on the introduction and use of OVFS, and promote the inclusion of OVFS documentation into the official OpenDAL documentation when the GSoC project is completed.

7 Project Timeline Schedule

This chapter provides an overview of the planning for the GSoC project. To ensure smooth progress and quality assurance, I've segmented the entire project lifecycle into several phases: planning and preparation phase, development phase, feedback and optimization phase, documentation improvements phase, and maintenance phase.

The specific objectives of each phase will be elaborated below, along with the phased deliverables expected to be completed at each phase, serving as checkpoints to assess the project's progress. Furthermore, upon the official launch of the project, overall development progress will be coordinated and updated through communication with mentors.

Planning And Preparation Phase

This phase encompasses the first week of the project and the period leading up to the official start of the GSoC project. In the early stages, it is crucial to throughly validate the feasibility of the project by enhancing understanding of Virtio and VirtioFS, and developing demo cases for verification testing. This lays a robust foundation for the subsequent phases of project development.

The preparation work in this phase can be roughly divided into two parts:

- Deepen understanding of the OpenDAL project and integrate into the open-source community environment. Continuously contribute to the OpenDAL project by resolving daily issues, thereby enhancing comprehension of the OpenDAL project's architecture and usage.
- Enhance programming skills related to VirtioFS by actively engaging in coding activities. Delve into the VirtioFS programming model based on Rust by examining the virtiofsd source code. Validate the handling of file system request interactions through hands-on demonstrations.

Development Phase

This phase signifies the pivotal stage of OVFS project development, requiring the advancement of project progress in alignment with the project planning and program design outlined in the proposal. The entire development phase can proceed based on three key objectives:

- Complete the end-to-end process verification of OVFS project based on local file systems. The functions implemented by the OVFS project at this stage should include: (1) Reading and writing data from the local file system on the host side through the OpenDAL project; (2) Providing a file system daemon that implements the vhost-user backend, enabling processing and interaction with file system requests within the VMs; (3) Supporting most file system interface semantics, such as data reading and writing, file creation and deletion, directory creation and deletion, based on the local file system.
- Utilize S3 as the distributed object storage target system to implement most of the functions provided by the OVFS project, and introduce configuration file support for the OVFS project. At this stage, the OVFS project has achieved the integration of file systems and object storage systems, enabling it to serve as storage backends based on various of storage systems.
- Optimize the code architecture of the OVFS project, implement asynchronous data reading and writing by introducing caching, and streamline the data access process. Fulfill other functionalities outlined in the proposal, such as DAX window and configuring hot restart, or incorporate new functionalities required by the OpenDAL community after consulting with mentors.

Furthermore, during the development process, attention should be given to supplementing test cases and ensuring the correct progression of development through thorough unit testing.

Feedback And Optimization Phase

At this phase, it's essential to regularly review the code with mentors, identify potential issues in the project development process, optimize logical organization and data structures, and ensure project quality through code optimization. This phase will recur multiple times throughout the project lifecycle and can serve as a summary of a development cycle. After achieving certain progress in development, the code should undergo review and optimization.

Documentation Improvements Phase

At this phase, most of the project's code work should have been completed. To better integrate the OVFS project into the OpenDAL project, attention needs to be given to supplementing OVFS project documentation and conducting more detailed testing of the project. Additionally, at this phase, it is necessary to commence preparation of the final acceptance materials for the GSoC project.

Maintenance Phase

This phase is a long-term endeavor, extending beyond just the last few weeks of the project. During this phase, it's crucial to maintain and flexibly develop and configure the the project according to the actual needs of the OpenDAL community. This includes developing related functionalities, building associated ecosystems, and continuously serving the community. The goal is to mature and stabilize the OVFS project and seamlessly integrate it into the OpenDAL project.

The specific plans and key time points of GSoC are outlined in the table below, with tasks corresponding to the description of each phase above:

Week	Date Range	Tasks
Week 1	05.27~06.02	Planning And Preparation Phase
Week 2	06.03~06.09	Development Phase 1: Achieving the first goal of the development phase
Week 3	06.10~06.16	Development Phase 1: Achieving the first goal of the development phase
Week 4	06.17~06.23	Feedback And Optimization Phase
Week 5	06.24~06.30	Development Phase 2: Achieving the second goal of the development phase
Week 6	07.01~07.07	Development Phase 2: Achieving the second goal of the development phase
Week 7	**07.08~07.14**	**submit midterm evaluations**
Week 8	07.15~07.21	Development Phase 3: Achieving all goals of the development phase
Week 9	07.22~07.28	Development Phase 3: Achieving all goals of the development phase
Week 10	07.29~08.04	Feedback and Optimization Phase
Week 11	08.05~08.11	Document Improvements Phase
Week 12	08.12~08.18	Maintenance Phase
Final Week	**08.19~08.26**	**submit final work product**

8 Why Me And Why Do I Wish To Take Part In GSoC 2024

I have many years of computer learning and programming experience, as well as many years of experience using the Rust language. I am familiar with the concepts related to file systems and have systematically studied the latest papers in the field of file systems. I have file system development experience. I developed a fully functional journaling flash file system using Rust, and optimized and tested the file system through tools such as perf, sysbench, and mdtest. I did some research and practice on file system metadata optimization in school. I have full confidence in realizing the OVFS project under the guidance of my mentors.

In the early stages of project application, I understood and learned the architecture and principles of OpenDAL by solving problems. Contact and communicate with the mentor through Discord and email to ensure the rationality of the proposal and OVFS project development. I make sure to have enough time to complete the GSoC project during project development. I plan to spend about 30 to 40 hours a week developing and enhancing the project. In addition, I will communicate with my mentor every week about the progress and challenges encountered during the project development process.

I very much hope to have the opportunity to participate in the GSoC 2024 project and implement the ideas and development of OVFS. I think GSoC is a very good learning opportunity and platform. By participating in this event, I can deeply participate in community activities and make my own contribution to the Apache OpenDAL community. I hope this opportunity can be the beginning of my participation in and contribution to the open source community. I also hope that I can participate more actively in the open source community and contribute my own strength in the future.

9 Contributions For OpenDAL

The chapter delineates the PRs that have been merged into the Apache OpenDAL project. As a newcomer, my current involvement in the OpenDAL community is limited. I aspire to persist in contributing and aiding the advancement of the OpenDAL community in the future.

feat(services/mongodb): add MongodbConfig

As part of RFC-3197, introduce a configuration structure for the MongoDB service to dynamically fetch service configurations at runtime. By exposing the MongoDB configuration to users, they are empowered to directly interact with the configuration structure.

feat(services/webdav): add jfrog test setup for webdav

OpenDAL supports access to storage services that offer WebDAV protocol. However, there are inherent differences in the implementation details among various storage services. JFrog Artifactory, being one such service supporting the WebDAV protocol, poses potential questions for OpenDAL as it hasn't been included in CI testing. This omission may lead to several issues when handling requests directed to JFrog Artifactory.

In PR, JFrog Artifactory test backend is added for WebDAV services based on docker and github actions. Through CI testing, it was found that OpenDAL's handling of the Webdav protocol was flawed. With the help of Xuanwo, related problems were fixed and the stable operation of OpenDAL in jfrog was ensured.

feat(services/swift): add support for storage_url configuration in swift service

When OpenDAL accesses the Swift service, it constructs the URL by concatenating fields such as endpoint and account. While this method works seamlessly with OpenStack Swift, it encounters difficulties accessing Swift services implemented using Ceph Rados Gateway.

This PR modified method by which OpenDAL constructs URLs when accessing the Swift service. URL construction is performed using the storage_url field returned during the authentication process of the Swift service. Relevant configuration and service codes have been updated accordingly, along with the service usage documentation.

feat(services/swift): add ceph test setup for swift

This PR introduces tests for Swift services using the Ceph Rados Gateway backend in github CI.

docs: update swift docs

This PR updated and remove the outdated description of Swift services usage by OpenDAL in official website.

docs: publish docs for object-store-opendal

object_store_opendal is an integration crate which uses OpenDAL as a backend for the object_store crate. This PR added a description to the official website for object_store_opendal crate, and built a CI to publish the crate's documents to official website through github actions.

fix(services/dropbox): fix dropbox batch test panic in ci

When dropbox performs batch deletion, dropbox will return an error type different from the official document API. OpenDAL will misinterpret this error and cause the tests in CI to fail. This PR adds support for error parsing in this scenario and fixes the problem.

docs: fix redirect error in object-store-opendal

This PR fixes a link pointing error in the official website.

docs(services/gcs): update gcs credentials description

This PR updates the description of the credential field and credential_path field in the GCS services configuration, making the configuration of these two fields clearer for users.

10 Reference

- https://opendal.apache.org/
- https://github.com/apache/opendal
- https://virtio-fs.gitlab.io/
- https://gitlab.com/virtio-fs/virtiofsd
- https://crates.io/crates/vm-memory
- https://crates.io/crates/virtio-queue
- https://crates.io/crates/vhost-user-backend
- https://www.qemu.org/
- https://katacontainers.io/
- https://github.com/hpc/ior
- https://github.com/axboe/fio
- https://github.com/akopytov/sysbench

- https://cloud.google.com/storage/docs/gcs-fuse
- https://groups.oasis-open.org/communities/tc-community-home2?CommunityKey=b3f5efa5-0e12-4320-873b-018dc7d3f25c