Mapping datasets to object storage

Aaron Chu, **Jeff LeFevre**, Carlos Maltzahn, Aldrin Montara, Peter Alvaro
University of California Santa Cruz

Dana Robinson, Quincey Koziol
HDF5 Group

CROSS | CENTER FOR RESEARCH IN OPEN SOURCE SOFTWARE

Agenda

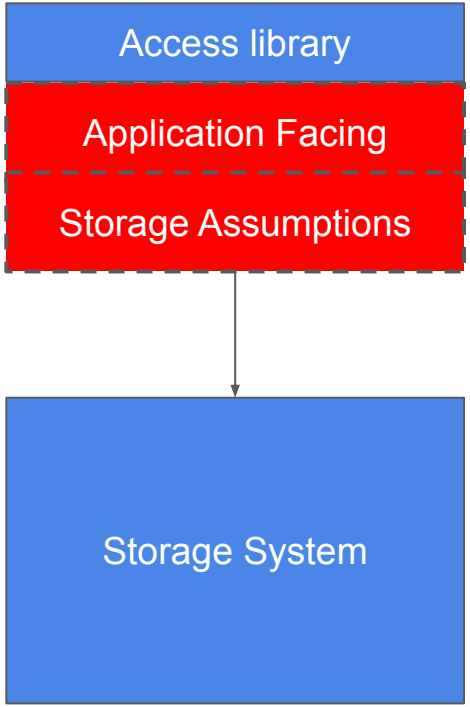
- Background and Problems
- Goals of the Project
- SkyhookDM Example
- HDF5 VOL Example

Background and Problems

- Scientists work with datasets.
- Datasets are mapped to storage systems via access libraries
- Access libraries make assumptions about storage systems.
- **Access libraries design assumptions are outdated (and will always be).**
- **Access libraries often do not scale out.**

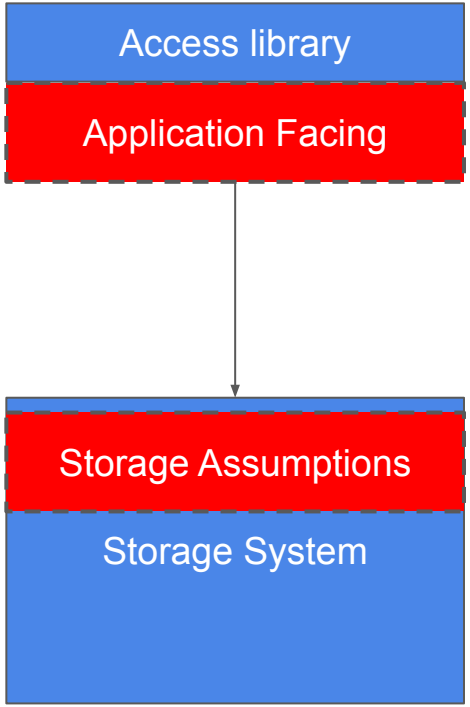
CENTER FOR RESEARCH IN OPEN SOURCE SOFTWARE

Goals of the project



- Currently 2 parts tied together
  - Application facing
  - Storage assumptions

Goals of the project



- Currently 2 parts tied together
  - Application facing
  - Storage assumptions

- Our Goals: break them apart
  - Make semantics of data available to storage system.
  - Allow the independent evolution of the access library and the backend storage systems.
  - Scale out the access library APIs and offload some operations to storage system.

Ceph distributed object storage

- Ceph is open source, highly scalable, widely available in the cloud
- Supports several APIs - file, S3, object-direct (librados)
- Object direct API allows users to write custom read/write methods
  - “OBJECT CLASSES” (CLS)
    - Read + filter
    - Write + compress
    - Write + filter + compress + create MD5 hash + (...)
    - Create thumbnails
- Scalable number of object storage servers (OSDs)
  - Each stores a collection of objects
- ***We develop custom methods through this interface***
  - Skyhook Data Management

SkyhookDM Example

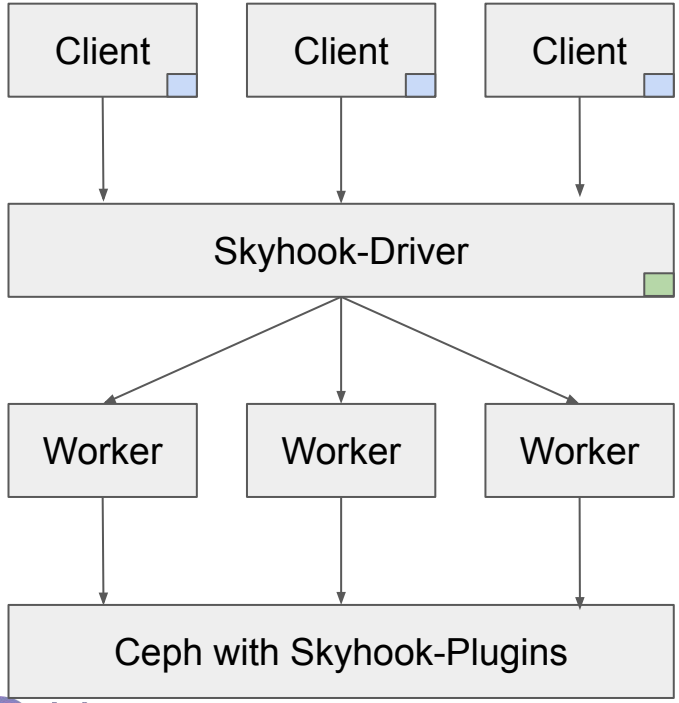
SkyhookDM with Ceph

- Data Partitioning and layout
  - physical data layout and format
  - fast in-memory serialization libraries (Google Flatbuffers, Apache Arrow)
  - data partitioning function
  - partition name to object name generation function

- Remote Processing
  - with our custom object classes
  - select, project, aggregate, compress,...

- Remote Indexing
  - locally query-able metadata (data-vals, stats)
  - stored in RocksDB on each storage server

SkyhookDM architecture



SkyhookDM Python Library using **Dask**

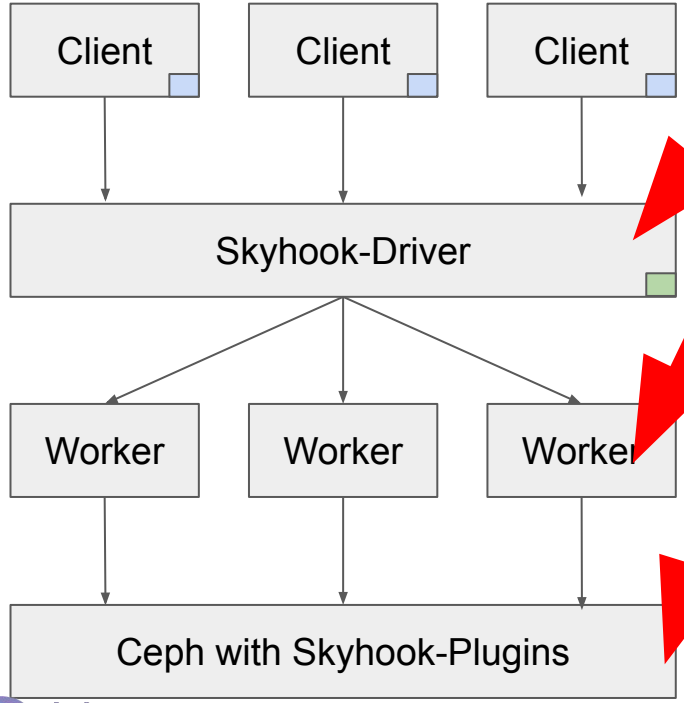
Skyhook Driver Python Library using **Dask**

Dask:

- **Dynamic task scheduling**
- **"Big Data" collections** (Dask Arrays, Dask Dataframe, Dask Bag, Dask Futures) run on top of dynamic task schedulers.

SkyhookDM architecture

SCALES INDEPENDENTLY!



SkyhookDM Python Library using **Dask**

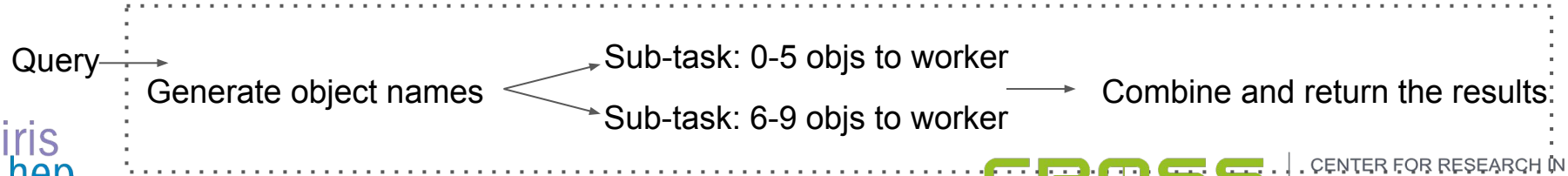
Skyhook Driver Python Library using **Dask**

Dask:

- **Dynamic task scheduling**
- **"Big Data" collections** (Dask Arrays, Dask Dataframe, Dask Bag, Dask Futures) run on top of dynamic task schedulers.

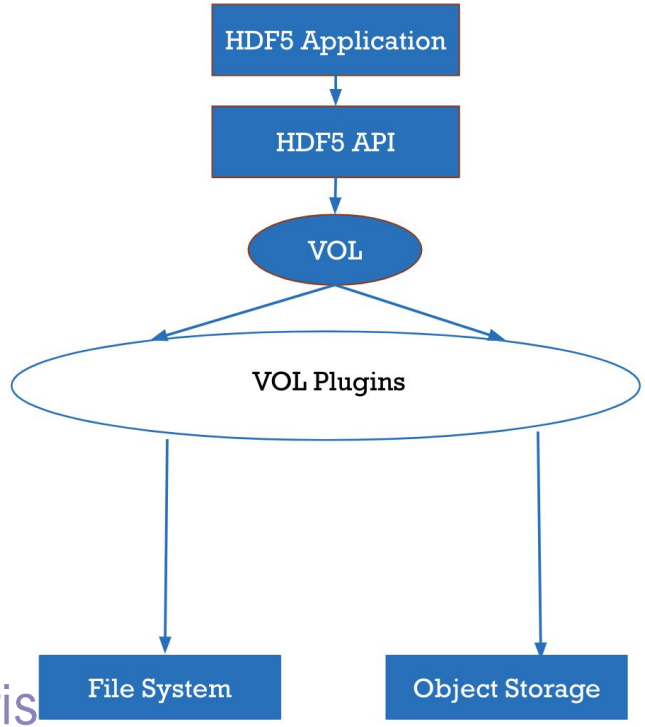
Example in Python

```
# Import SkyhookDM
from skyhook import SkyhookDM

# Create a SkyhookDM Object
sk = SkyhookDM()

# Connect to the Skyhook Driver
sk.connect('10.10.1.2')

# Run a Query
data = sk.query('mydataset', ['project price,tax', 'select price>5'])

# Get the table schema
sk .getSchema('mydataset')
```



HDF5 VOL Example

Virtual Object Layer (VOL) for HDF5



Access libraries are separated from the implementation of the storage system.

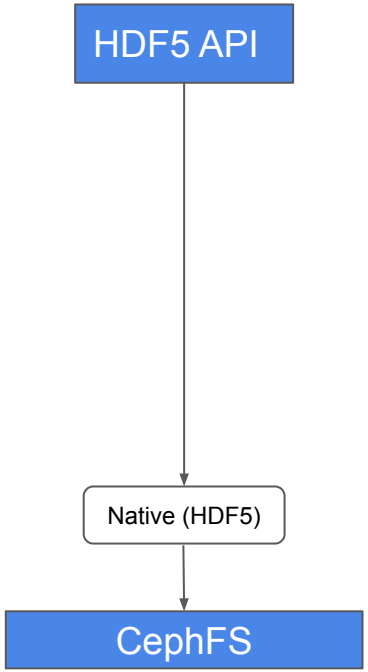
Access libraries and backend storages can evolve independently.

Allow access libraries to map onto themselves.

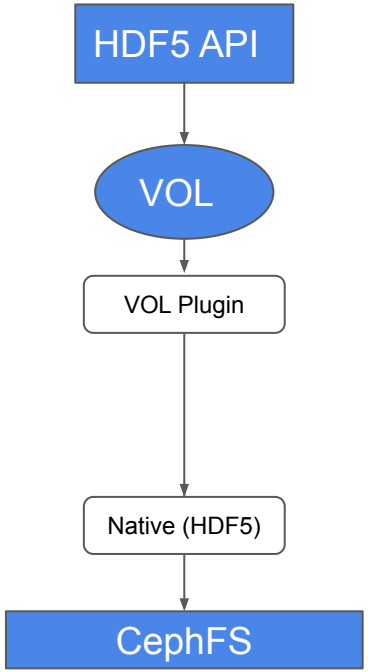
Allow distribution and offloading the access operations across multiple servers.

Allow global and local optimizations.

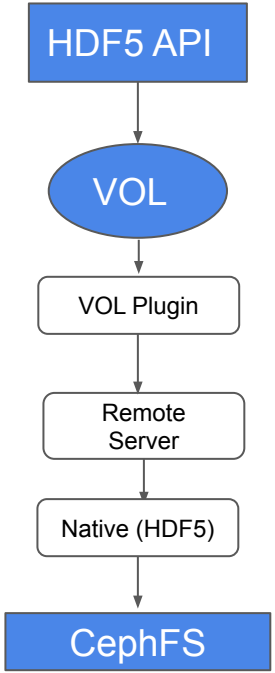
VOL Plugin Models



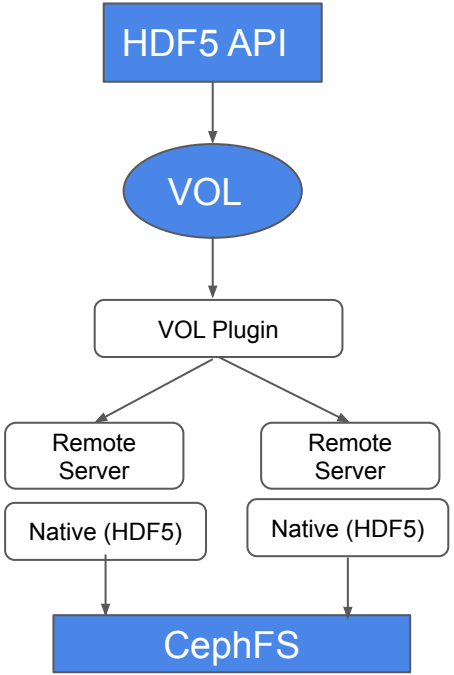
No VOL



Local VOL

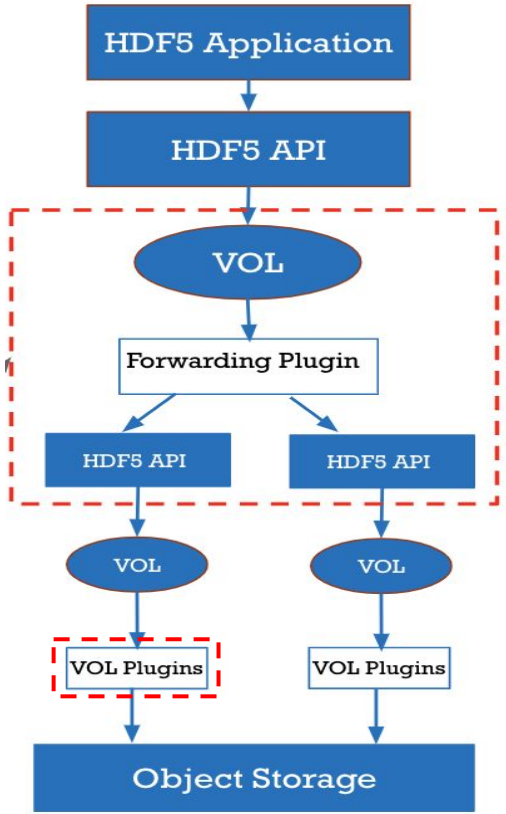


Remote VOL



Distributed VOL

The VOL Example



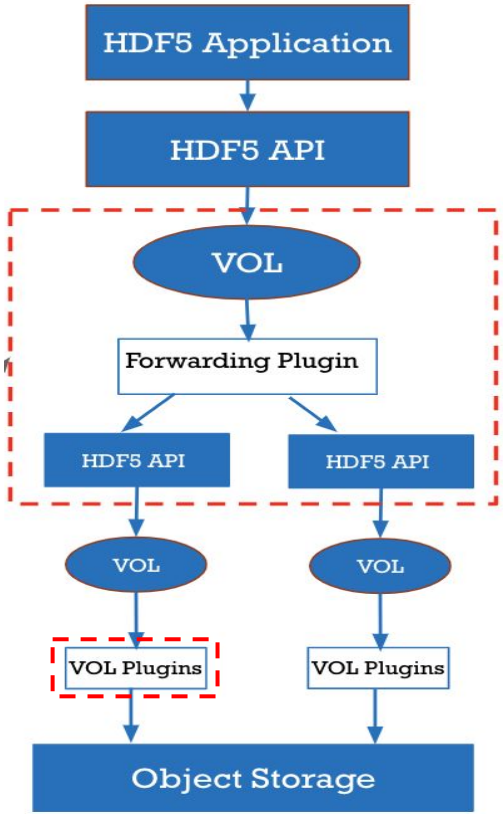
Forwarding Plugin

- Intelligent Data Partition Policy
- Efficient framework to manage distributed tasks, e.g. Ray or Dask.

Local Plugins

- Build an additional VOL plugin to do the local optimization on each storage server.

The VOL Example



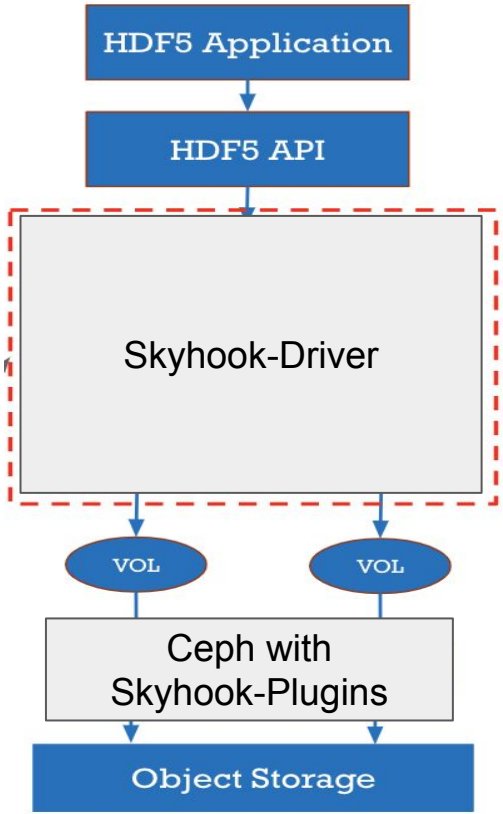
Forwarding Plugin

- Intelligent Data Partition Policy
- Efficient framework to manage distributed tasks, e.g. Ray or Dask.

Local Plugins

- Build an additional VOL plugin to do the local optimization on each storage server.
- **Different VOL plugins can capture the characteristics of local storage server (SW/HW).**

The VOL Example - tying the models together...



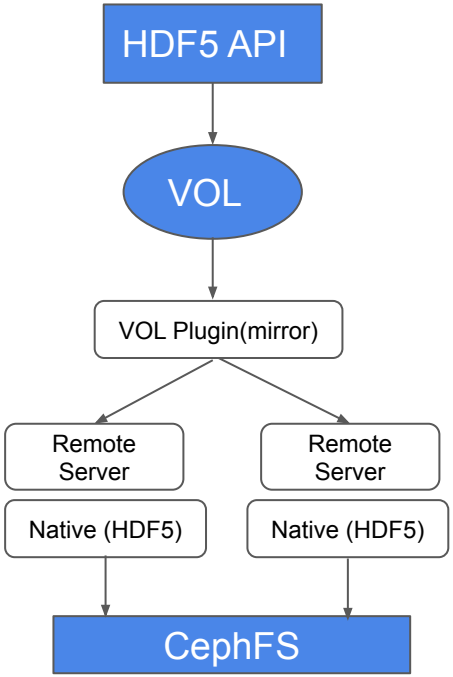
Forwarding Plugin

- Intelligent Data Partition Policy
- Efficient framework to manage distributed tasks, e.g. Ray or Dask.

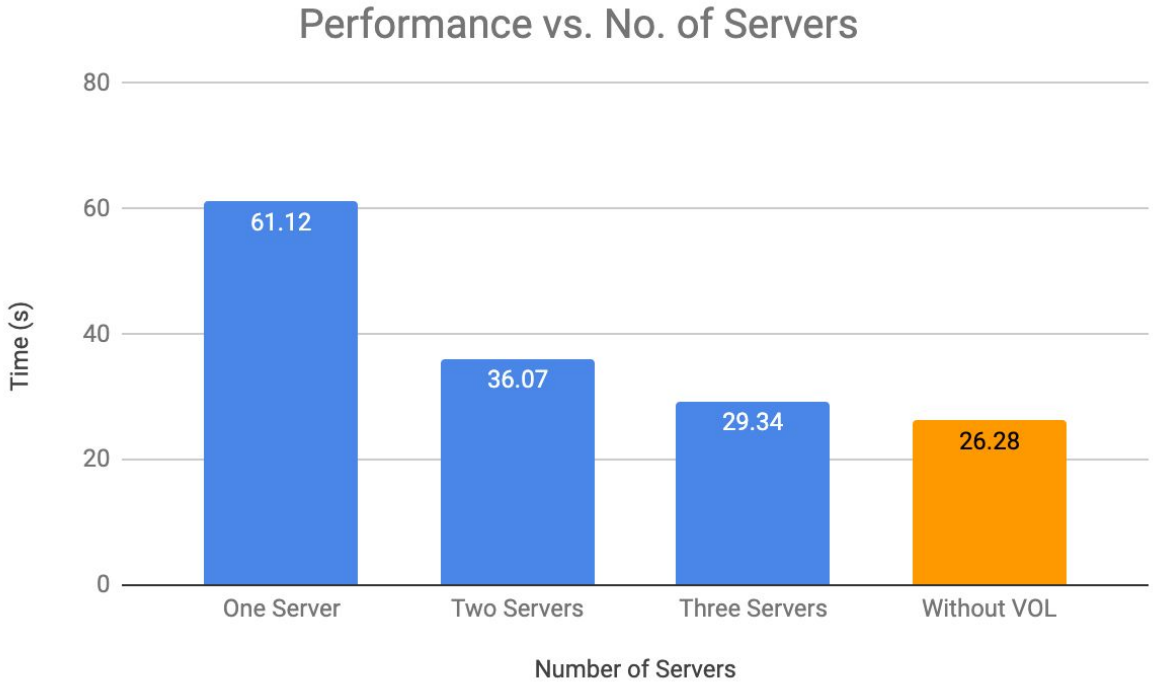
Local Plugins

- Build an additional VOL plugin to do the local optimization.
- **Different VOL plugins can capture the characteristics of local storage server (SW/HW).**

Distribute the writing (3GB Data)



Distributed VOL



At least 3 storage servers are needed to offset the overhead of distributed VOL.

Conclusion and ongoing work

- SkyhookDM model addresses scale out of data access using existing interfaces
  - Scientific file libraries - that have “VOL” interface
    - Maps onto themselves
  - Databases - that have External Table access mechanism
    - Maps to external data sources

- Overhead of VOL (or external tables) is non-zero
  - Can amortize with scalability of distributed object storage systems

- In-progress
  - Mapping ROOT files to objects (branches/events), use uproot library
  - Custom partition policies for common usage scenarios
  - Format/deliver partitions as Arrow tables

Thank You!

xweichu@ucsc.edu
jlefevre@ucsc.edu

Funding: IRIS-HEP (NSF-OAC 1836650), UCSC Center for Research in Open Source Software

CROSS | CENTER FOR RESEARCH IN OPEN SOURCE SOFTWARE