HSDS Serverless Support with AWS Lambda and Direct Access

John Readey

The HDF Group

Overview

- HDF as a Service – HSDS
- HSDS Storage Model
- HSDS Serverless with AWS Lambda
- HSDS Serverless with Direct Access

HDF as a Service - HSDS

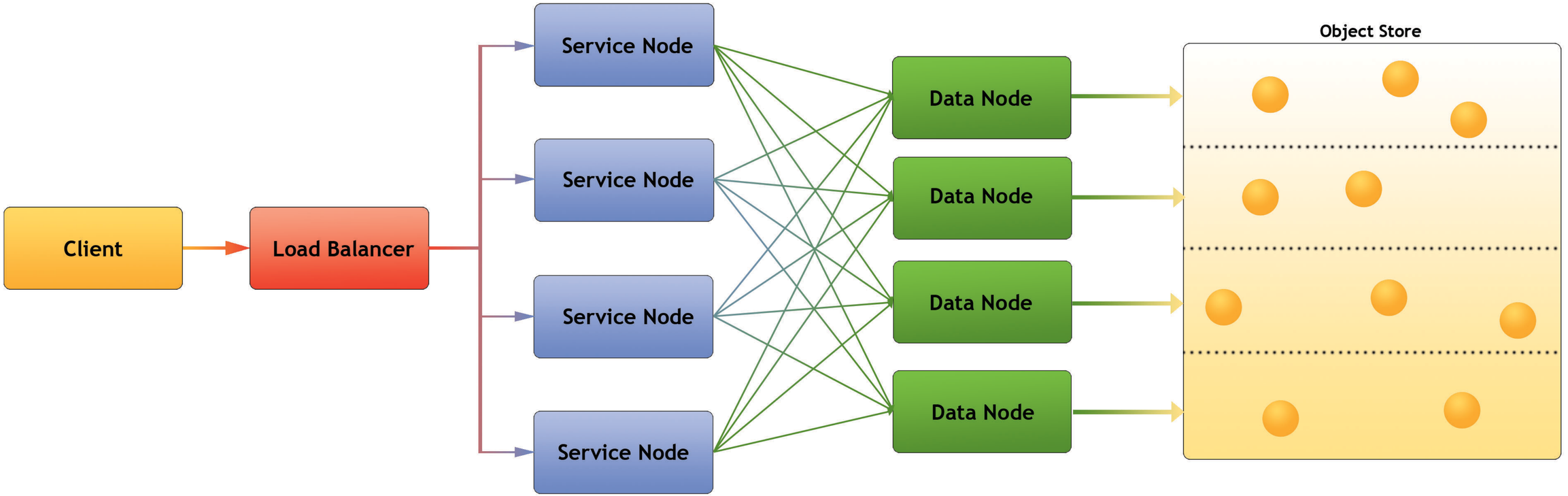
HSDS – Highly Scalable Data Service -- is a REST-based web service for HDF data

Design criteria:

- Performant – good to great performance
- Scalable – Run across multiple cores and/or clusters
- Feature complete – Support (most) of the features provided by the HDF5 library
- Utilize POSIX or object storage (e.g. AWS S3, Azure Blob Storage)

Note: HSDS was originally developed as a NASA ACCESS 2015 project: https://earthdata.nasa.gov/esds/competitive-programs/access/hsds

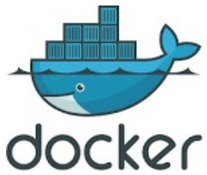
HSDS Architecture



- Client: Any user of the service
- Load balancer – distributes requests to Service nodes
- Service Nodes – processes requests from clients (with help from Data Nodes)
- Data Nodes – responsible for partition of Object Store
- Object Store: Base storage service (e.g. AWS S3)

HSDS Platforms

HSDS is implemented as a set of containers and can be run on common container management systems:



Using different supported storage systems:

POSIX
Filesystem

Why an HDF Service?

There are several benefits to enabling HDF as a Service…

- Allow remote access to large datasets (the inertia of big data)
- Coordinate actions of multiple reader/writer clients
- Provide language-neutral interface to HDF
  - Though h5pyd (Python) and REST VOL (C/C++) provide HDF5 API compatibility
- Enable web-based applications
- Facilitate container-based applications (Docker, Kubernetes, Mesos)

HSDS Storage Model

- Instead of managing HDF5 objects (datasets, groups, chunks) within a POSIX file, HSDS stores each HDF object as a separate file (or object within an object storage system such as S3)
  - For meta data (datasets and groups), content is saved in a self-descriptive manner using JSON
  - For chunks, data is stored as a binary object for efficiency
- Storing larger datasets in this manner will lead to a large number of objects to store one “file”, but HDF API and tools abstract user from needing to deal with the underlying storage system
- The HSDS Storage model is ”cloud compatible” since it allows efficient updates when object storage systems (e.g. S3, Azure Blob Storage, Ceph) are used

Why a sharded data format?

- Limit maximum size of any object
  - -> Object storage systems typically don’t support partial writes, so large objects are inefficient to update
- Supporting parallelism is easier
  - -> no file locking needed
- No need to manage free space, key-value mappings, etc
  - -> storage systems have gotten pretty good at doing this for you
- No need to worry about system crash leaving you with corrupted HDF5 files
  - -> worse case you lose one object, with object storage not even that

HSDS sharded schema example


```
root_obj_id/
    group.json
    obj1_id/
        group.json
    obj2_id/
        dataset.json
        0_0
        0_1
    obj3_id/
        dataset.json
        0_0_2
        0_0_3
```

Observations:

- Metadata is stored as JSON
- Chunk data stored as binary blobs
- Self-explanatory
- One HDF5 file can translate to lots of objects
- Flat hierarchy – supports HDF5 multilinking
- Can limit maximum size of an object
- Can be used with Posix or object storage

Schema is documented here:
https://github.com/HDFGroup/hsds/blob/master/docs/design/obj_store_schema/obj_store_schema_v2.md

HSDS sans Service

- The next HSDS release will provide two new (though related) capabilities:
  - AWS Lambda support - HSDS implemented as a Lambda Function
  - Direct Access Model – HSDS implemented entirely on the client
- Both of these enable developers to take advantage of HSDS capabilities without the need to run a server

Pros and cons of running a service

- Accessing a sharded data store via a service (HSDS) is great:
  - Server mediates access to the storage system
  - Server can speed things up by caching recently accessed data
  - Minimizes data I/O between client & server (e.g. remote clients)
  - HSDS running on a large server or cluster can provide more processing capacity and memory than a client might have
  - HSDS serves as a synchronization point for multi-writer/multi-reader algorithms
- Unless it's not:
  - Don't want to bother setting up, running service
  - Challenge to scale capacity of service to clients
  - Cloud charges for running server 24/7

AWS Lambda

- AWS Lambda is an AWS service that enables function to be executed on demand without the need to provision any infrastructure
- Price (with 1GB memory) is: $0.0000000167/ms
- Beyond potential cost savings, Lambda enables “infinite elasticity”
  - Can support widely varying workloads without need to spin up/down servers
  - By default, 1000 instances of a function can be run simultaneously
- No infrastructure management required
  - i.e. os patches, server backup, monitoring, etc.

AWS Lambda Challenges

- Adapting existing container-based server to run on Lambda is not trivial…
  - Software needs to initialize as quickly as possible (to minimize latency)
  - Unable to take advantage of caching
  - Limited to max 6 VCPUs per function
  - Lambda runtime environment is restricted:
    - No equivalent to docker to manage multiple containers
    - TCP not allowed (which is how HSDS containers talk to each other)
    - Shared memory not allowed

HSDS Lambda Design

- To minimize the need for special purpose code, HSDS is implemented on Lambda as follows:
  - On startup SN node and DN nodes are run as sub-processes (vs containers)
    - Number of DN nodes based on available VCPUs
    - Nodes communicate via Unix Domain sockets (vs TCP)
  - Payload is unpacked to a HTTP request and sent to the SN node
  - SN node distributes work over DN nodes
  - DN nodes read/write to S3, return result to SN node
  - Response is returned as the Lambda result

HSDS Lambda Performance Constraints

- Compared with HSDS running on a dedicated server, the response time will be 2-100x slower
- Performance challenges:
  - 2-4 seconds for a ”cold” function to startup
  - ~0.5 seconds for HSDS code to initialize
  - All data must be fetched from S3 (no cache to utilize)
  - Limited number of cores (number of DN nodes) available

HSDS Lambda Performance Results

- Simple performance test:
  - (10000, 1602, 2976) dimension data, hyperslab selection of [:, x, y]
  - 400 chunks access (~1GB data)
  - 10,000 elements returned (~39KB)
- Results
  - Server HSDS w/ 4 nodes: 5.04s, 204 MB/s
  - Lambda HSDS w/ 1GB mem: 25.6s (cold), 40 MB/s
  - Lambda HSDS w/ 1GB mem: 18.1s (warm), 56 MB/s
  - Lambda HSDS w/ 10GB mem: 10.2s (warm), 102 MB/s

HSDS Direct Access

- HSDS Direct Access enables client code to incorporate HSDS functionality without the need of a server
- As with HSDS Lambda, enables “serverless” operation
- Direct Access is best used for applications that run “close” to the storage and don’t require close multi-client synchronization

HSDS Direct Access Architecture

As with HSDS Lambda:

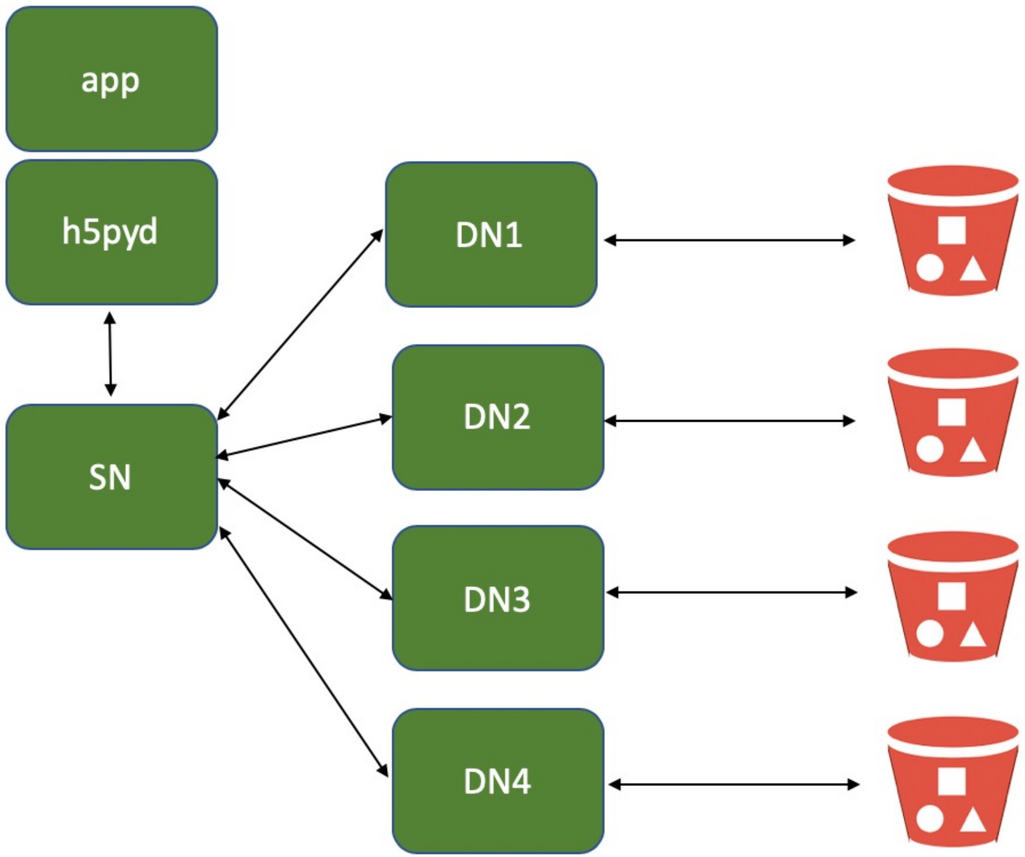
- SN code would run in a sub-process
- DN code would run in one or more sub-processes (e.g. based on number of cores)
- Communication between parent processes and sub-processes would be via Unix Domain sockets
- Sub-processes shutdown when last file is closed

But all code executes on local system

The Python H5PYD package will initiate direct access when endpoint is specified as ‘local’.

Otherwise the application will function in same manner as with server

Direct Access Architecture



Number of DN nodes can be set to number of cores

All green boxes run as processes on client system

S3, Azure Blob, or Posix storage

Direct Access VOL plugin

- For C/C++ apps, the direct access model could be implemented as a VOL connector
- Other than launching the sub-processes the VOL would work in the same way as the REST VOL, so it probably makes sense to include this functionality in the REST VOL rather than create a new VOL
- With direct access HDF5 lib + REST VOL enables sharded data as an alternative to the HDF5 file format
  - Enables multi-threading (& parallelization support for even single threaded apps)
  - Enables async API
  - Cloud optimized storage (or Posix)
  - Crash-proof

Release Target

- AWS Lambda support is available now in the HSDS 0.7.0 beta release
  - Available on github
- Official release of HSDS 0.7.0 should be available this summer
- HSDS Direct access support will be available in h5pyd v0.8.0
  - (also planned for summer 2021)
- Feedback (github issues, PRs, email) is encouraged

Questions?

Try it out!

Get the software here:

- HSDS: https://github.com/HDFGroup/hsds
- H5pyd: https://github.com/HDFGroup/h5pyd
- REST VOL: https://github.com/HDFGroup/vol-rest
- REST API documentation: https://github.com/HDFGroup/hdf-rest-api
- Example programs: https://github.com/HDFGroup/hdflab_examples