HDF Data Services

HDF5 in the Cloud

The HDF Group

John Readey
The HDF Group
jreadey@hdfgroup.org

My Background

The HDF Group

Sr. Architect at The HDF Group

Started in 2014

Have been exploring remote interfaces to HDF

Previously: Dev Manager at Amazon/AWS

amazon web services

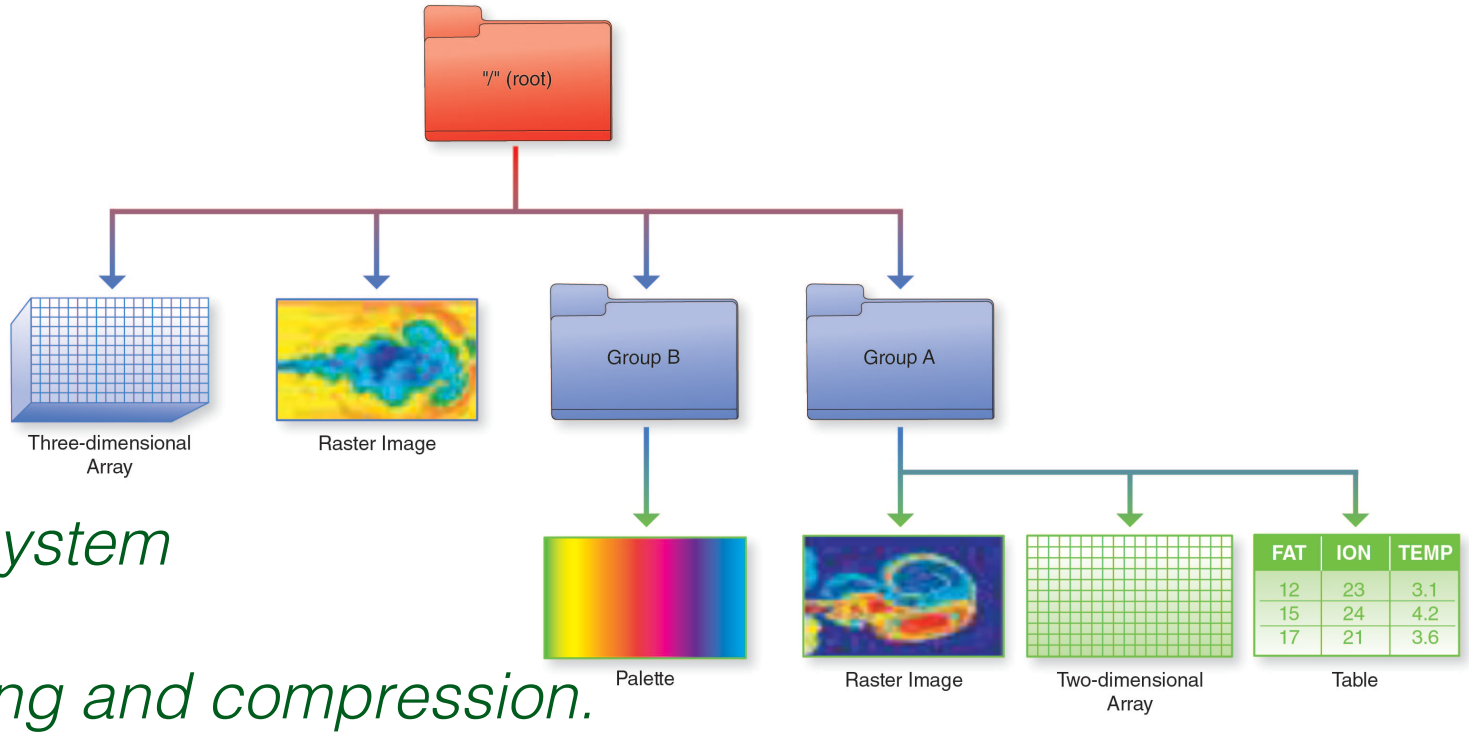
More previously: Used HDF5 while a developer at Intel

intel inside

What is HDF5?

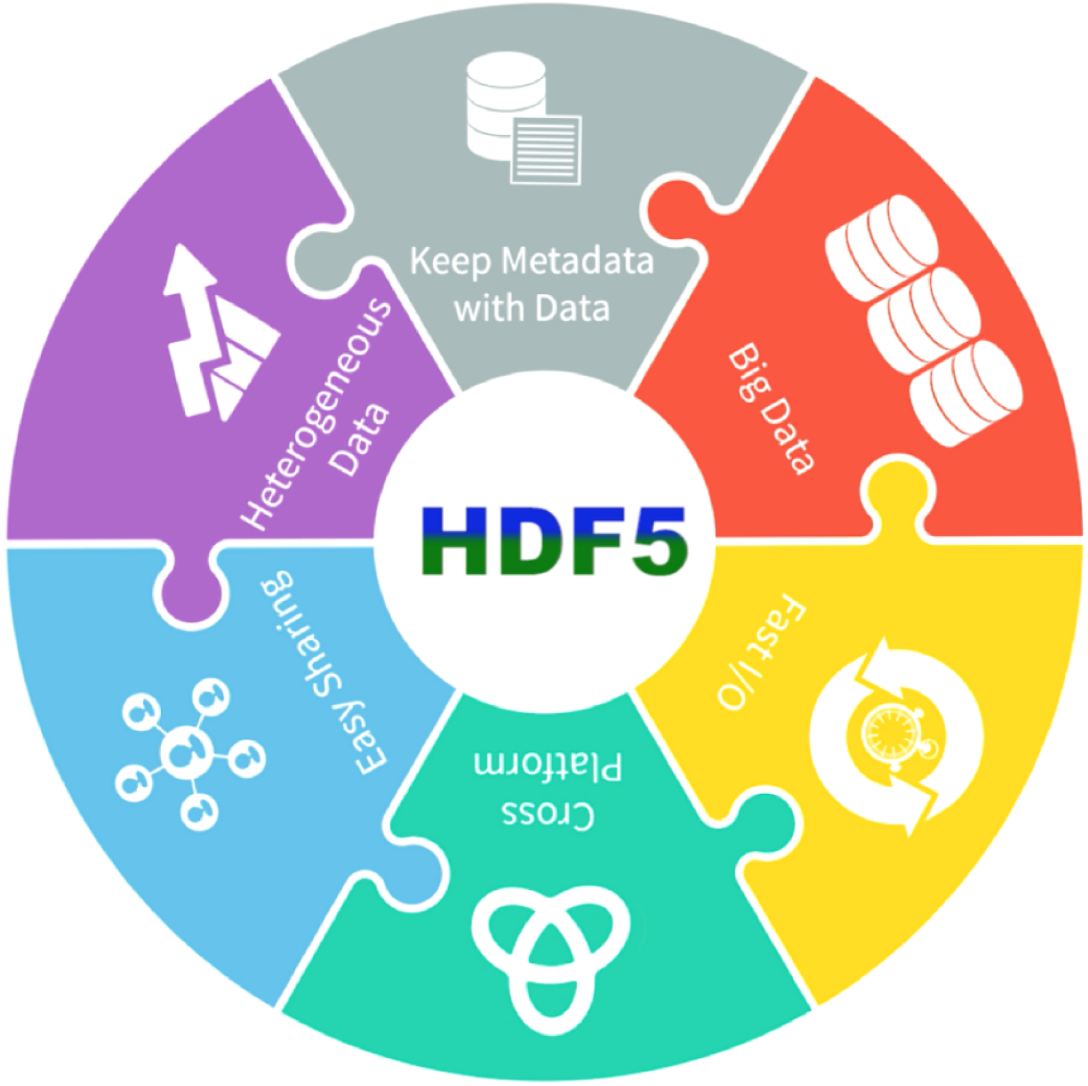
Depends on your point of view:

- **a C-API**
- **a File Format**
- **a data model**



Think of HDF5 as a file system within a file.
Store arrays with chunking and compression.
Add NumPy style data selection.

Why is this concept so different + useful?



- **Native support for multidimensional data**
- **Data and metadata in one place => streamlines data lifecycle & pipelines**
- Portable, no vendor lock-in
- Maintains logical view while adapting to storage context
- In-memory, over-the-wire, on-disk, parallel FS, object store
- Pluggable filter pipeline for compression, checksum, encryption, etc.
- High-performance I/O
- Large ecosystem (700+ Github projects)

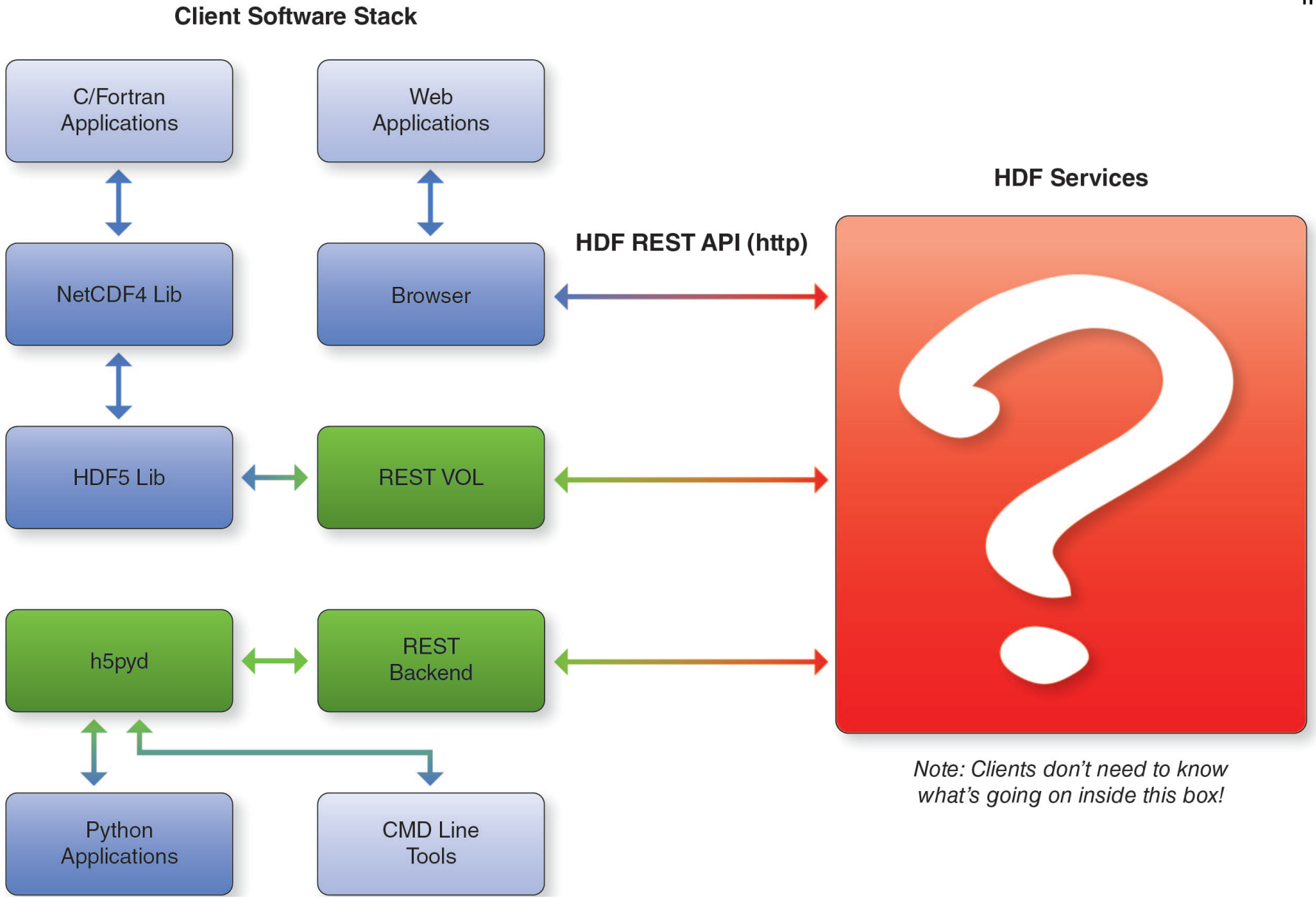
Introducing Highly Scalable Data Service (HSDS)

- HDF5 optimized for the cloud
- Storage using AWS S3
  - Built in redundancy
  - Cost effective
  - Scalable throughput
- Runs as a cluster of Docker containers
  - Elastically scale compute with usage
- Feature compatible with HDF5 library
- Implemented in Python using asyncio
  - Task oriented parallelism

HSDS Features

- Clients can interact with service using REST API
- SDKs provide language specific interface (e.g. h5pyd for Python)
- Can read/write just the data they need (as opposed to transferring entire files)
- No limit to the amount of data that can be stored by the service
- Multiple clients can read/write to same data source
- Scalable performance:
  - Can cache recently accessed data in RAM
  - Can parallelize requests across multiple nodes
  - More nodes -> better performance

Client/Server Architecture



HSDS S3 Schema

How to store HDF5 content in S3?

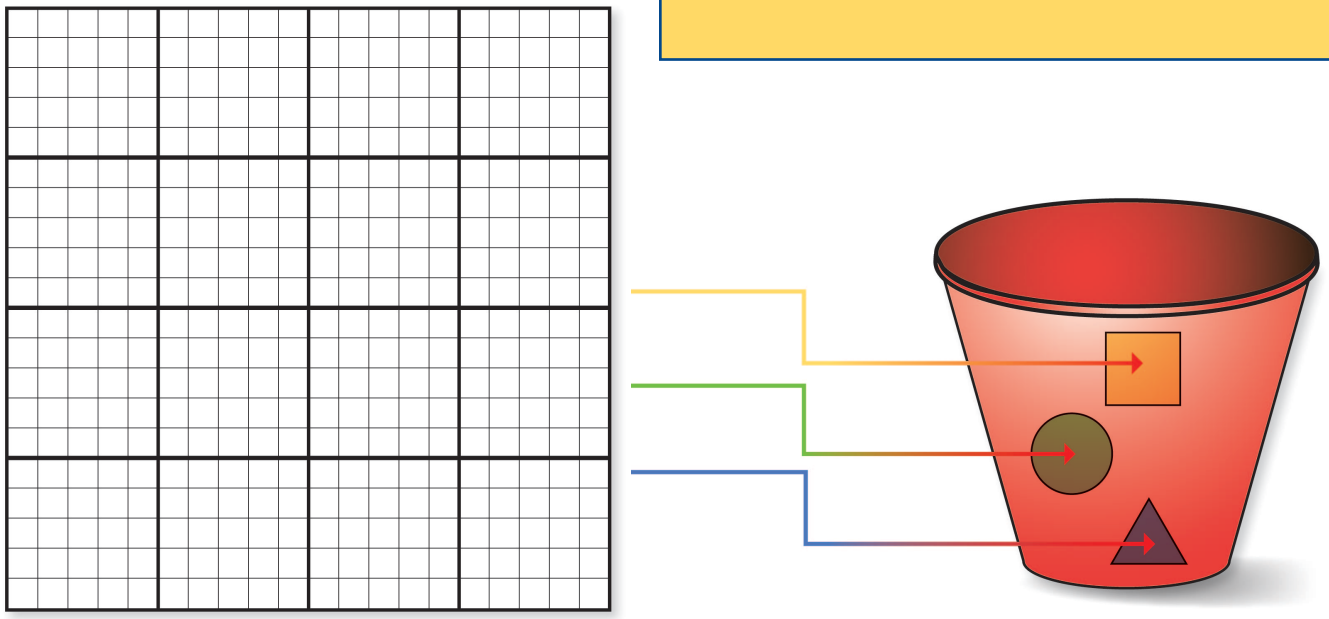
- Limit maximum storage object size
- Support parallelism for read/write
- Only data that is modified needs to be updated
- (Potentially) Multiple clients can be reading/updating the same “file”

Big Idea: Map individual HDF5 objects (datasets, groups, chunks) as Object Storage Objects

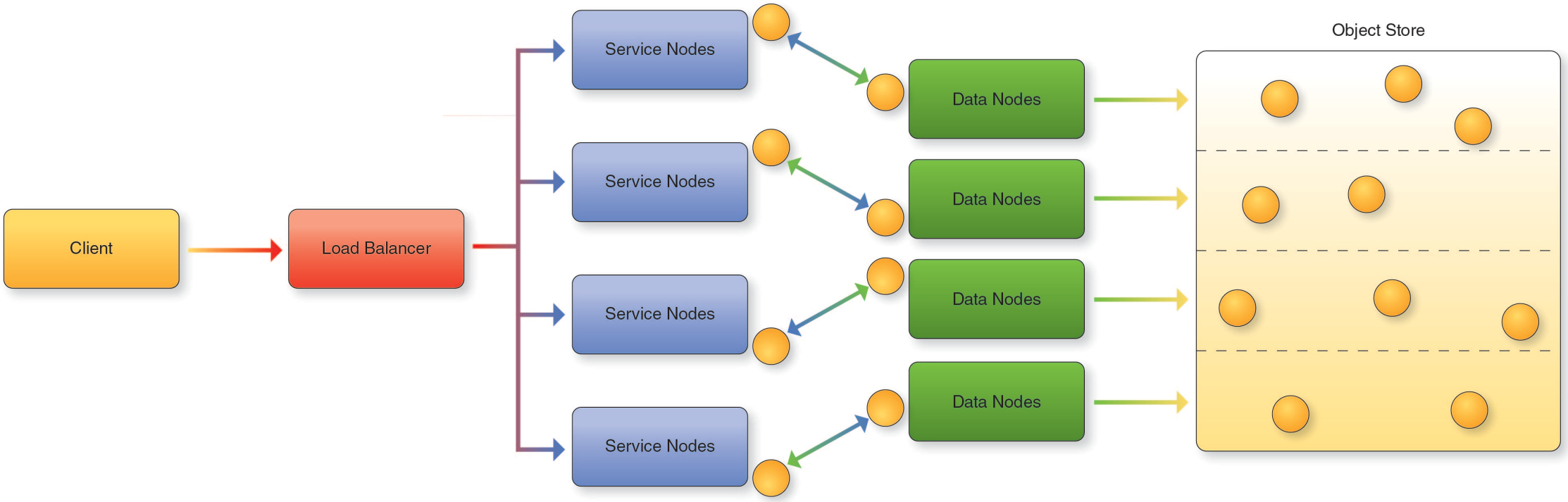
Each chunk (heavy outlines) get persisted as a separate object

Legend:

- *Dataset is partitioned into chunks*
- *Each chunk stored as an S3 object*
- *Dataset meta data (type, shape, attributes, etc.) stored in a separate object (as JSON text)*



Architecture for HSDS



Legend:

- 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)

Implementing HSDS with asyncio

- **HSDS relies heavily on Python's new asyncio module**
  - **Concurrency based on *tasks* (rather than say multithreading or multiprocessing)**
  - **Task switching occurs when process would otherwise wait on I/O**

```
async def my_func():
    a_regular_function_call()
    await a_blocking_call()
```

- Control will switch to another task when await is encountered
- Result is the app can do other useful work vs. blocking
- Supporting 1000's of concurrent tasks within a process is quite feasible

Parallelizing data access with asyncio

- **SN node invoking parallel requests on DN nodes**

```
tasks = []
for chunk_id in my_chunk_list:
    task = asyncio.ensure_future(read_chunk_query(chunk_id))
    tasks.append(task)
await asyncio.gather(*tasks, loop=loop)
```

- Read_chunk_query makes a http request to a specific DN node
- Set of DN nodes can be reading from S3, decompression and selecting requested data in parallel
- Asyncio.gather waits for all tasks to complete before continuing
- Meanwhile, new requests can be processed by SN node

Python and Docker

- Docker makes developing clustered applications sooo much easier
  - Can run dozens of containers on a moderate laptop
  - Containers communicate with each other just like on a physical network
  - Use docker stats to check up cpu, net i/o, disk i/o usage per container
  - Can try out different constraints for amount of memory, disk per container
  - Same code “just works” on an actual cluster
  - ”scale up” by launching more containers on production hardware
  - AWS ECS enables running containers in a machine agnostic way
- Using docker does require a reversion to the edit/build/run paradigm
  - The build step is now the creation of the docker image
  - Run is launching the container(s)

Python package MVPs

- numpy – python arrays
  - Used heavily in server and client stacks
  - Great performance for common array operations
  - Simplifies much of the logic needed for hyperslab selection
- aiohttp – async http client/server
  - Use of asyncio requires async enabled packages
  - Aiohttp is used in HSDS as both web server and client
- Aiobotocore – async aws s3 client
  - Enables async read/write to S3
- H5py – template for h5pyd package

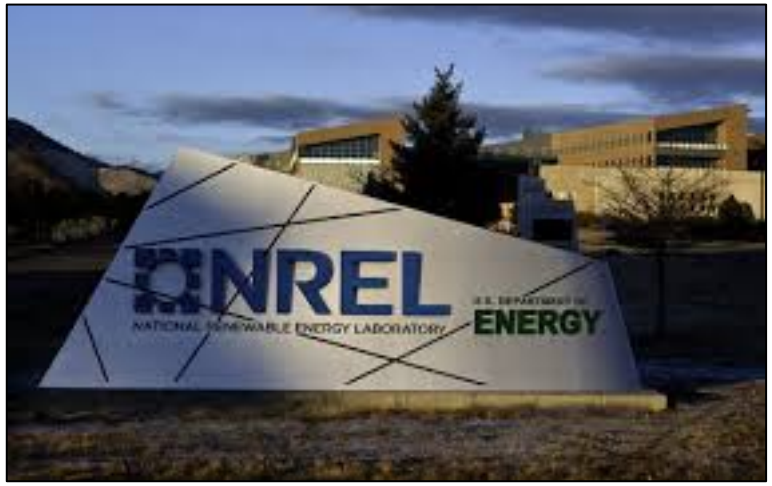
H5pyd – Python client for HDF Server

- H5py is a popular Python package that provide a Pythonic interface to the HDF5 library
- H5pyd (for h5py distributed) provides a h5py compatible h5py for accessing the server
- Pure Python – uses requests package to make http calls to server
- Compatible with h5serv (the reference implementation of the HDF REST API)
- Include several extensions to h5py:
  - List content in folders
  - Get/Set ACLs (access control list)
  - Pytables-like query interface

HSDS CLI (Command Line Interface)

- Accessing HDF via a service means can't utilize usual shell commands: ls, rm, chmod, etc.
- Command line tools are a set of simple apps to use instead:
  - hsinfo: display server version, connect info
  - hsls: list content of folder or file
  - hstouch: create folder or file
  - hsdel: delete a file
  - hsload: upload an HDF5 file
  - hsget: download content from server to an HDF5 file
  - hsacl: create/list/update ACLs (Access Control Lists)
- Implemented in Python & uses h5pyd

Demo Time!



NREL (National Renewable Energy Laboratory) is using HSDS to make 7TB of wind simulation data accessible to the public.

Datasets are three-dimensional covering the continental US:

- Time (one slice/hour)
- Lon (~2k resolution)
- Lat (~2k resolution)

Initial data covers one year (8760 slices), but will be soon be extended to 5 years (35 TBs).

Rather than downloading TB's of files, interested users can now use the HSDS client libraries to explore the datasets.

Future Work

- Work planned for the next year
  - Compression
  - Variable length datatypes
  - NetCDF support
  - Auto Scaling
  - Scalability and performance testing

To Find out More:

- H5serv: https://github.com/HDFGroup/h5serv
- Documentation: http://h5serv.readthedocs.io/
- H5pyd: https://github.com/HDFGroup/h5pyd
- RESTful HDF5 White Paper: https://www.hdfgroup.org/pubs/papers/RESTful_HDF5.pdf
- Blog articles:
  - https://hdfgroup.org/wp/2015/04/hdf5-for-the-web-hdf-server/
  - https://hdfgroup.org/wp/2015/12/serve-protect-web-security-hdf5/
  - https://www.hdfgroup.org/2017/04/the-gfed-analysis-tool-an-hdf-server-implementation/

HDF5 Community Support

- Documentation - https://support.hdfgroup.org/documentation/
  - Tutorials, FAQs, examples
- HDF-Forum – mailing list and archive
  - Great for specific questions
- Helpdesk Email – help@hdfgroup.org
  - Issues with software and documentation

https://support.hdfgroup.org/services/community_support.html

Questions? Comments?

The HDF Group

www.hdfgroup.org

Dave Pearah
CEO
David.Pearah@hdfgroup.org

Dax Rodriguez
Director of Commercial Services and Solutions
Dax.Rodriguez@hdfgroup.org