EOSDIS
NASA'S EARTH OBSERVING SYSTEM
DATA AND INFORMATION SYSTEM

# OPeNDAP Clients, Aggregation and S3

Summer ESIP, 2019

James Gallagher
EED-2 Contractor
*jgallagher@opendap.org*

Nathan Potter
EED-2 Contractor
*ndp@opendap.org*

Kodi Neumiller
EED-2 Contractor
*kneumiller@opendap.org*

This work was supported by NASA/GSFC under Raytheon Co. contract number NNG15HZ39C.

## Background

- Five client applications have been tested with Hyrax serving data stored on Amazon's S3 Web Object Store.
- We tested:
    - Access to data from a single file
    - Access to data from aggregations of multiple files
- Two kinds of aggregations were tested:
    - Aggregations using NcML*
    - Aggregations using the 'virtual sharding' technique we have developed for use with S3
- Exciting bonus material...

*NetCDF Markup Language

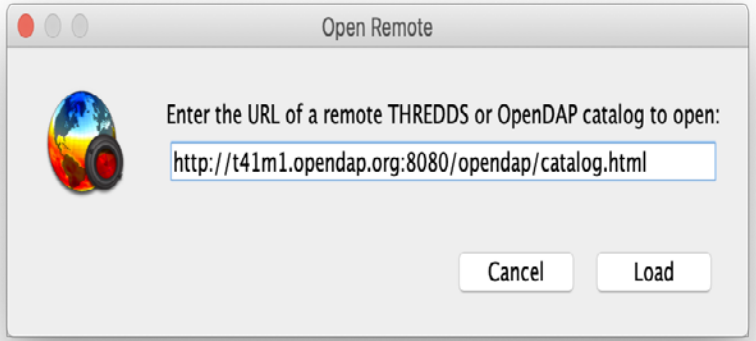
## The Five Clients

1. Panoply – a Java client; built-in knowledge of DAP[1] and THREDDS[2] catalogs, uses the Java netCDF library
2. Jupyter notebooks & xarray – Python (can use PyDAP or netCDF C/Python)
3. NCO – a C client, C netCDF library
4. ArcGIS – a C (or C++?) client, either libdap or C netCDF (we're not sure)
5. GDAL – a C++ client, libdap

[1]Data Access Protocol, [2]Thematic Realtime Environmental Distributed Data Services

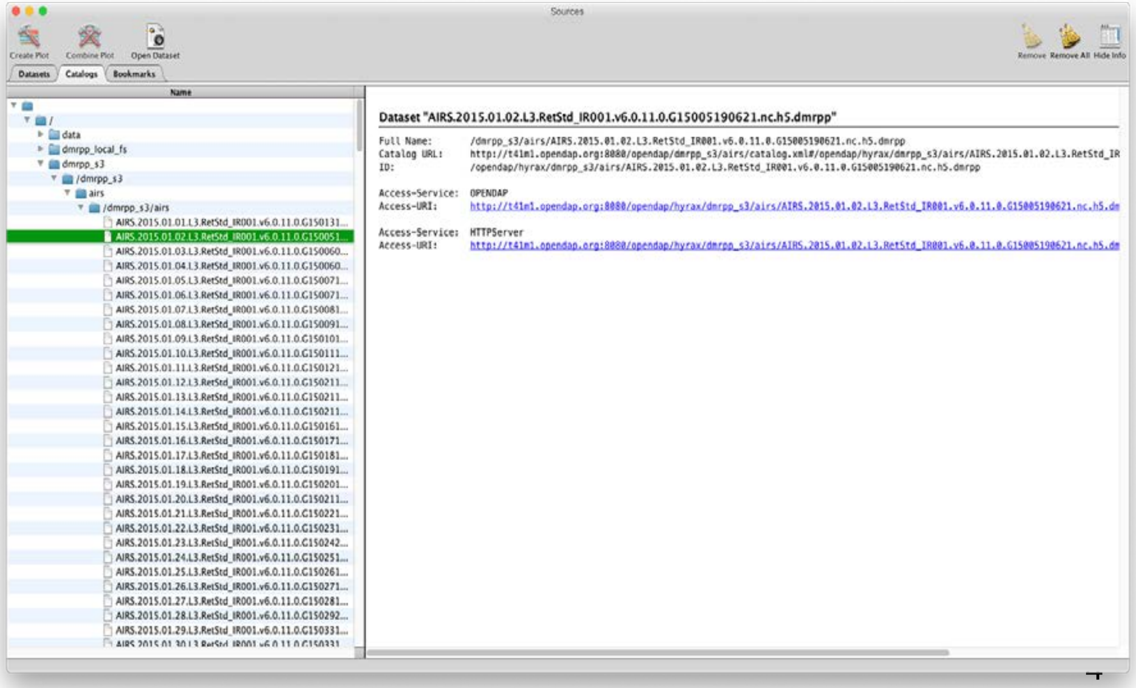
# Panoply

- See live demo (using 4.0.5, which has some fixes for servers that use Tomcat 8 – nothing to do with DAP or S3)
- To open a server's catalog: File-->Open Remote Catalog...
    - http://t41m1.opendap.org:8080/opendap/catalog.html



Enter the URL of a remote THREDDS or OpenDAP catalog to open:

http://t41m1.opendap.org:8080/opendap/catalog.html



```
▼ /
    ▶ data
    ▶ dmrpp_local_fs
    ▼ dmrpp_s3
        ▼ /dmrpp_s3
            ▼ airs
                ▼ /dmrpp_s3/airs
                    AIRS.2015.01.01.L3.RetStd_IR001.v6.0.11.0.G150131...
                    AIRS.2015.01.02.L3.RetStd_IR001.v6.0.11.0.G150051...
                    AIRS.2015.01.03.L3.RetStd_IR001.v6.0.11.0.G150060...
                    AIRS.2015.01.04.L3.RetStd_IR001.v6.0.11.0.G150060...
                    AIRS.2015.01.05.L3.RetStd_IR001.v6.0.11.0.G150071...
                    AIRS.2015.01.06.L3.RetStd_IR001.v6.0.11.0.G150071...
                    AIRS.2015.01.07.L3.RetStd_IR001.v6.0.11.0.G150081...
                    AIRS.2015.01.08.L3.RetStd_IR001.v6.0.11.0.G150091...
                    AIRS.2015.01.09.L3.RetStd_IR001.v6.0.11.0.G150101...
                    AIRS.2015.01.10.L3.RetStd_IR001.v6.0.11.0.G150111...
                    AIRS.2015.01.11.L3.RetStd_IR001.v6.0.11.0.G150121...
                    AIRS.2015.01.12.L3.RetStd_IR001.v6.0.11.0.G150211...
                    AIRS.2015.01.13.L3.RetStd_IR001.v6.0.11.0.G150211...
                    AIRS.2015.01.14.L3.RetStd_IR001.v6.0.11.0.G150211...
                    AIRS.2015.01.15.L3.RetStd_IR001.v6.0.11.0.G150161...
                    AIRS.2015.01.16.L3.RetStd_IR001.v6.0.11.0.G150171...
                    AIRS.2015.01.17.L3.RetStd_IR001.v6.0.11.0.G150181...
                    AIRS.2015.01.18.L3.RetStd_IR001.v6.0.11.0.G150191...
                    AIRS.2015.01.19.L3.RetStd_IR001.v6.0.11.0.G150201...
                    AIRS.2015.01.20.L3.RetStd_IR001.v6.0.11.0.G150211...
                    AIRS.2015.01.21.L3.RetStd_IR001.v6.0.11.0.G150221...
                    AIRS.2015.01.22.L3.RetStd_IR001.v6.0.11.0.G150231...
                    AIRS.2015.01.23.L3.RetStd_IR001.v6.0.11.0.G150242...
                    AIRS.2015.01.24.L3.RetStd_IR001.v6.0.11.0.G150251...
                    AIRS.2015.01.25.L3.RetStd_IR001.v6.0.11.0.G150261...
                    AIRS.2015.01.26.L3.RetStd_IR001.v6.0.11.0.G150271...
                    AIRS.2015.01.27.L3.RetStd_IR001.v6.0.11.0.G150281...
                    AIRS.2015.01.28.L3.RetStd_IR001.v6.0.11.0.G150292...
                    AIRS.2015.01.29.L3.RetStd_IR001.v6.0.11.0.G150331...
                    AIRS.2015.01.30.L3.RetStd_IR001.v6.0.11.0.G150331
```

**Dataset "AIRS.2015.01.02.L3.RetStd_IR001.v6.0.11.0.G15005190621.nc.h5.dmrpp"**

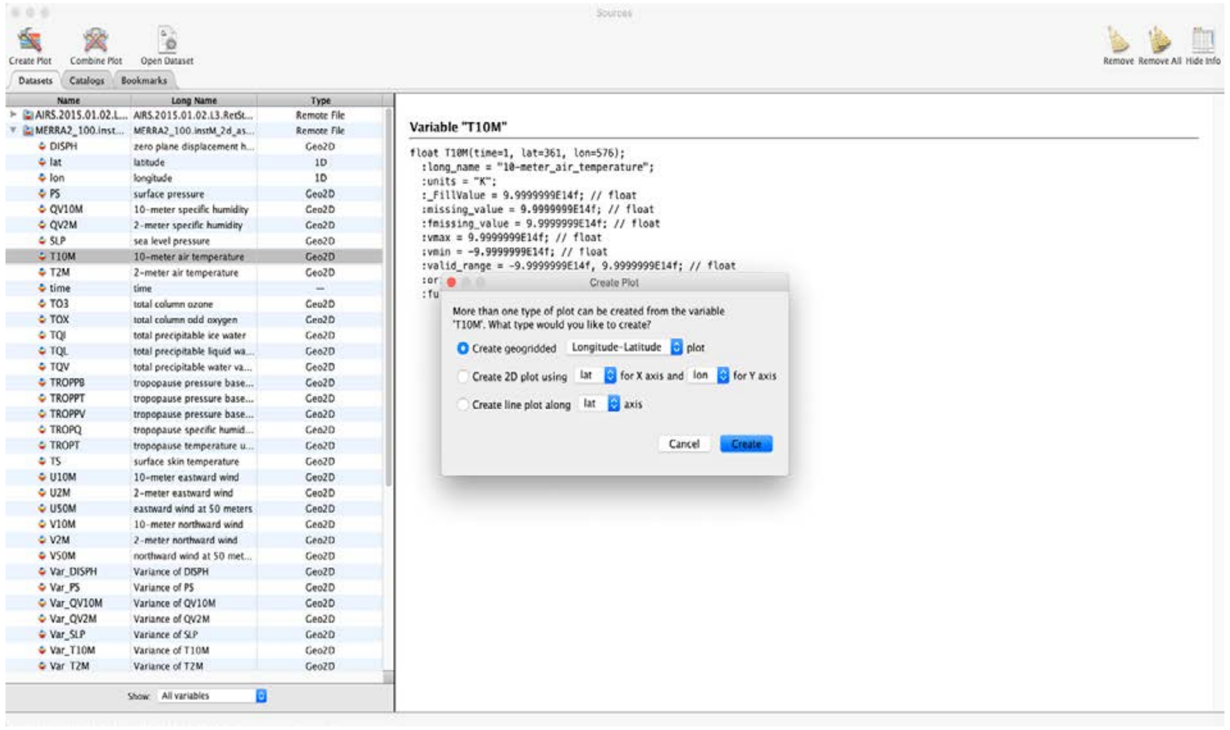
```
Full Name:       /dmrpp_s3/airs/AIRS.2015.01.02.L3.RetStd_IR001.v6.0.11.0.G15005190621.nc.h5.dmrpp
Catalog URL:     http://t41m1.opendap.org:8080/opendap/dmrpp_s3/airs/catalog.xml#/opendap/hyrax/dmrpp_s3/airs/AIRS.2015.01.02.L3.RetStd_IR
ID:              /opendap/hyrax/dmrpp_s3/airs/AIRS.2015.01.02.L3.RetStd_IR001.v6.0.11.0.G15005190621.nc.h5.dmrpp

Access-Service:  OPENDAP
Access-URI:      http://t41m1.opendap.org:8080/opendap/hyrax/dmrpp_s3/airs/AIRS.2015.01.02.L3.RetStd_IR001.v6.0.11.0.G15005190621.nc.h5.dm

Access-Service:  HTTPServer
Access-URI:      http://t41m1.opendap.org:8080/opendap/hyrax/dmrpp_s3/airs/AIRS.2015.01.02.L3.RetStd_IR001.v6.0.11.0.G15005190621.nc.h5.dm
```

# Panoply, continued

- To open a single dataset directly: File-->Open Remote Dataset...
    - http://t41m1.opendap.org:8080/opendap/dmrpp_s3/merra2/MERRA2_100.instM_2d_asm_Nx.198001.nc4.dmrpp



| Name | Long Name | Type |
|---|---|---|
| AIRS.2015.01.02.L... | AIRS.2015.01.02.L3.RetSt... | Remote File |
| MERRA2_100.inst... | MERRA2_100.instM_2d_as... | Remote File |
| DISPH | zero plane displacement h... | Geo2D |
| lat | latitude | 1D |
| lon | longitude | 1D |
| PS | surface pressure | Geo2D |
| QV10M | 10-meter specific humidity | Geo2D |
| QV2M | 2-meter specific humidity | Geo2D |
| SLP | sea level pressure | Geo2D |
| T10M | 10-meter air temperature | Geo2D |
| T2M | 2-meter air temperature | Geo2D |
| time | time | — |
| TO3 | total column ozone | Geo2D |
| TOX | total column odd oxygen | Geo2D |
| TQI | total precipitable ice water | Geo2D |
| TQL | total precipitable liquid wa... | Geo2D |
| TQV | total precipitable water va... | Geo2D |
| TROPPB | tropopause pressure base... | Geo2D |
| TROPPT | tropopause pressure base... | Geo2D |
| TROPPV | tropopause pressure base... | Geo2D |
| TROPQ | tropopause specific humid... | Geo2D |
| TROPT | tropopause temperature u... | Geo2D |
| TS | surface skin temperature | Geo2D |
| U10M | 10-meter eastward wind | Geo2D |
| U2M | 2-meter eastward wind | Geo2D |
| U50M | eastward wind at 50 meters | Geo2D |
| V10M | 10-meter northward wind | Geo2D |
| V2M | 2-meter northward wind | Geo2D |
| V50M | northward wind at 50 met... | Geo2D |
| Var_DISPH | Variance of DISPH | Geo2D |
| Var_PS | Variance of PS | Geo2D |
| Var_QV10M | Variance of QV10M | Geo2D |
| Var_QV2M | Variance of QV2M | Geo2D |
| Var_SLP | Variance of SLP | Geo2D |
| Var_T10M | Variance of T10M | Geo2D |
| Var_T2M | Variance of T2M | Geo2D |

Show: All variables

**Variable "T10M"**

```
float T10M(time=1, lat=361, lon=576);
  :long_name = "10-meter_air_temperature";
  :units = "K";
  :_FillValue = 9.9999999E14f; // float
  :missing_value = 9.9999999E14f; // float
  :fmissing_value = 9.9999999E14f; // float
  :vmax = 9.9999999E14f; // float
  :vmin = -9.9999999E14f; // float
  :valid_range = -9.9999999E14f, 9.9999999E14f; // float
  :or
  :fu
```

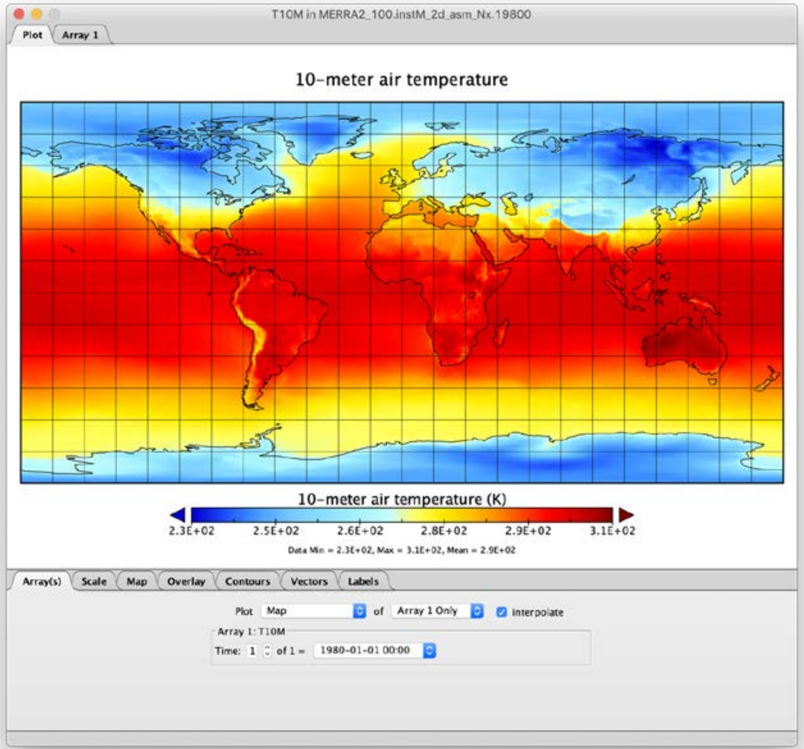
Create Plot

More than one type of plot can be created from the variable 'T10M'. What type would you like to create?

[✓ Create geogridded] Longitude-Latitude plot

Create 2D plot using lat for X axis and lon for Y axis

Create line plot along lat axis



Data Min = 2.3E+02, Max = 3.1E+02, Mean = 2.9E+02

Plot Map of Array 1 Only [✓ interpolate]

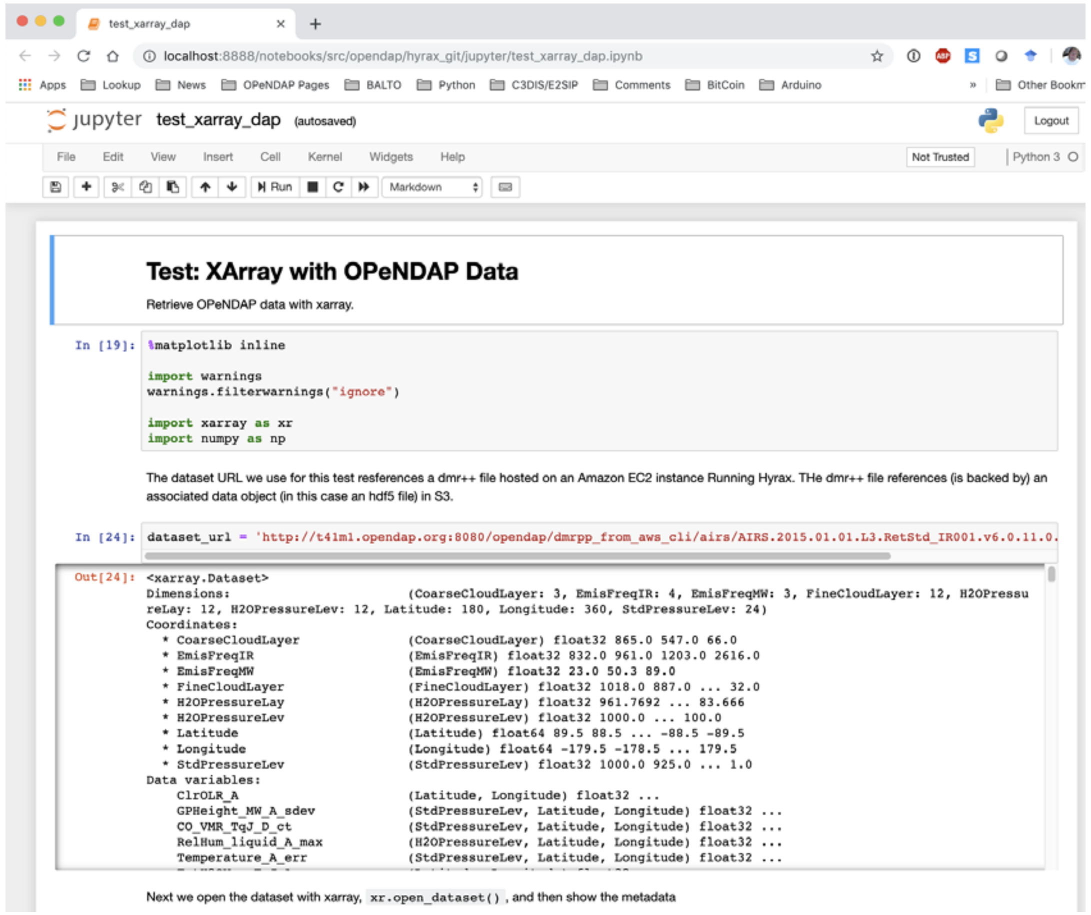
Array 1: T10M

Time: 1 of 1 = 1980-01-01 00:00

# Jupyter notebooks and xarray

## Download the notebook from the Jira ticket (HK-380)

Use dataset_url = 'http://t41m1.opendap.org:8080/opendap/dmrpp_s3/airs/AIRS.2015.01.01.L3.RetStd_IR001.v6.0.11.0.G15013155825.nc.h5.dmrpp'



**Test: XArray with OPeNDAP Data**

Retrieve OPeNDAP data with xarray.

```
In [19]: %matplotlib inline

         import warnings
         warnings.filterwarnings("ignore")

         import xarray as xr
         import numpy as np
```

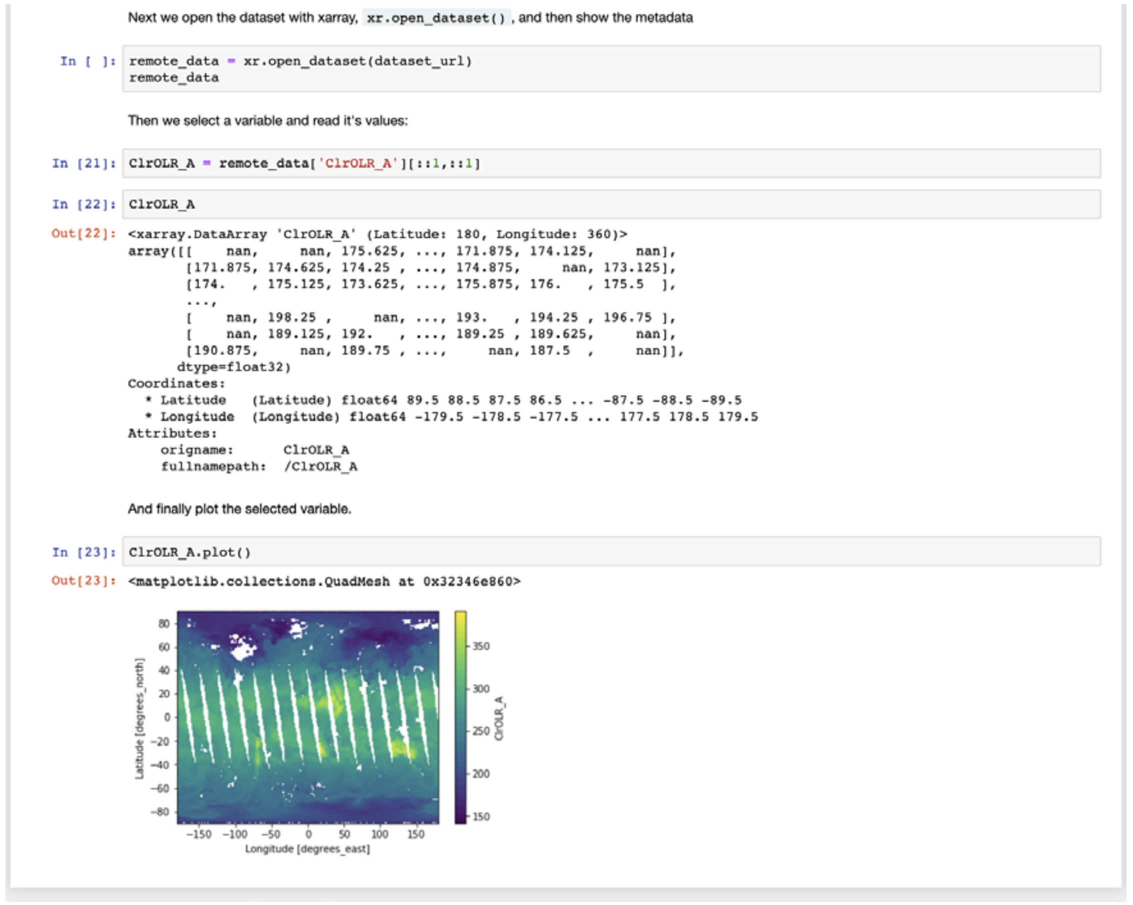
The dataset URL we use for this test resferences a dmr++ file hosted on an Amazon EC2 instance Running Hyrax. THe dmr++ file references (is backed by) an associated data object (in this case an hdf5 file) in S3.

```
In [24]: dataset_url = 'http://t41m1.opendap.org:8080/opendap/dmrpp_from_aws_cli/airs/AIRS.2015.01.01.L3.RetStd_IR001.v6.0.11.0.

Out[24]: <xarray.Dataset>
         Dimensions:                  (CoarseCloudLayer: 3, EmisFreqIR: 4, EmisFreqMW: 3, FineCloudLayer: 12, H2OPressu
         reLay: 12, H2OPressureLev: 12, Latitude: 180, Longitude: 360, StdPressureLev: 24)
         Coordinates:
           * CoarseCloudLayer     (CoarseCloudLayer) float32 865.0 547.0 66.0
           * EmisFreqIR           (EmisFreqIR) float32 832.0 961.0 1203.0 2616.0
           * EmisFreqMW           (EmisFreqMW) float32 23.0 50.3 89.0
           * FineCloudLayer       (FineCloudLayer) float32 1018.0 887.0 ... 32.0
           * H2OPressureLay       (H2OPressureLay) float32 961.7692 ... 83.666
           * H2OPressureLev       (H2OPressureLev) float32 1000.0 ... 100.0
           * Latitude             (Latitude) float64 89.5 88.5 ... -88.5 -89.5
           * Longitude            (Longitude) float64 -179.5 -178.5 ... 179.5
           * StdPressureLev       (StdPressureLev) float32 1000.0 925.0 ... 1.0
         Data variables:
             ClrOLR_A             (Latitude, Longitude) float32 ...
             GPHeight_MW_A_sdev   (StdPressureLev, Latitude, Longitude) float32 ...
             CO_VMR_TqJ_D_ct      (StdPressureLev, Latitude, Longitude) float32 ...
             RelHum_liquid_A_max  (H2OPressureLev, Latitude, Longitude) float32 ...
             Temperature_A_err    (StdPressureLev, Latitude, Longitude) float32 ...
```

Next we open the dataset with xarray, `xr.open_dataset()`, and then show the metadata

...



Next we open the dataset with xarray, `xr.open_dataset()`, and then show the metadata

```
In [ ]: remote_data = xr.open_dataset(dataset_url)
        remote_data
```

Then we select a variable and read it's values:

```
In [21]: ClrOLR_A = remote_data['ClrOLR_A'][::1,::1]

In [22]: ClrOLR_A

Out[22]: <xarray.DataArray 'ClrOLR_A' (Latitude: 180, Longitude: 360)>
         array([[     nan,      nan, 175.625, ..., 171.875, 174.125,      nan],
                [171.875, 174.625, 174.25 , ..., 174.875,      nan, 173.125],
                [174.   , 175.125, 173.625, ..., 175.875, 176.   , 175.5  ],
                ...,
                [     nan, 198.25 ,      nan, ..., 193.   , 194.25 , 196.75 ],
                [     nan, 189.125, 192.   , ..., 189.25 , 189.625,      nan],
                [190.875,      nan, 189.75 , ...,      nan, 187.5  ,      nan]],
               dtype=float32)
         Coordinates:
           * Latitude   (Latitude) float64 89.5 88.5 87.5 86.5 ... -87.5 -88.5 -89.5
           * Longitude  (Longitude) float64 -179.5 -178.5 -177.5 ... 177.5 178.5 179.5
         Attributes:
             origname:      ClrOLR_A
             fullnamepath:  /ClrOLR_A
```

And finally plot the selected variable.

```
In [23]: ClrOLR_A.plot()

Out[23]: <matplotlib.collections.QuadMesh at 0x32346e860>
```

## Roundup: NCO, ArcGIS, GDAL

- All of these work the same when data are stored on S3
- There are some quirks for each,
  - NCO: A command-line tool, this is easy to run in the cloud and thus an easy way to 'move compute to the data.'
  - ArcGIS/ArcMAP: Has a special 'OPeNDAP Raster' option
  - GDAL: To build DAP access, must be built with the '--with-dods-root' option
- Some common issues:
  - Companies may be using an older version of the netCDF library that does not work with Tomcat 8.5. Lobby them to upgrade.
  - Any client can be run 'in the cloud;' for GUI[1] clients, use VNC[2].
  - The configuration for VNC is not trivial

[1]Graphical User Interface, [2]Virtual Network Computing

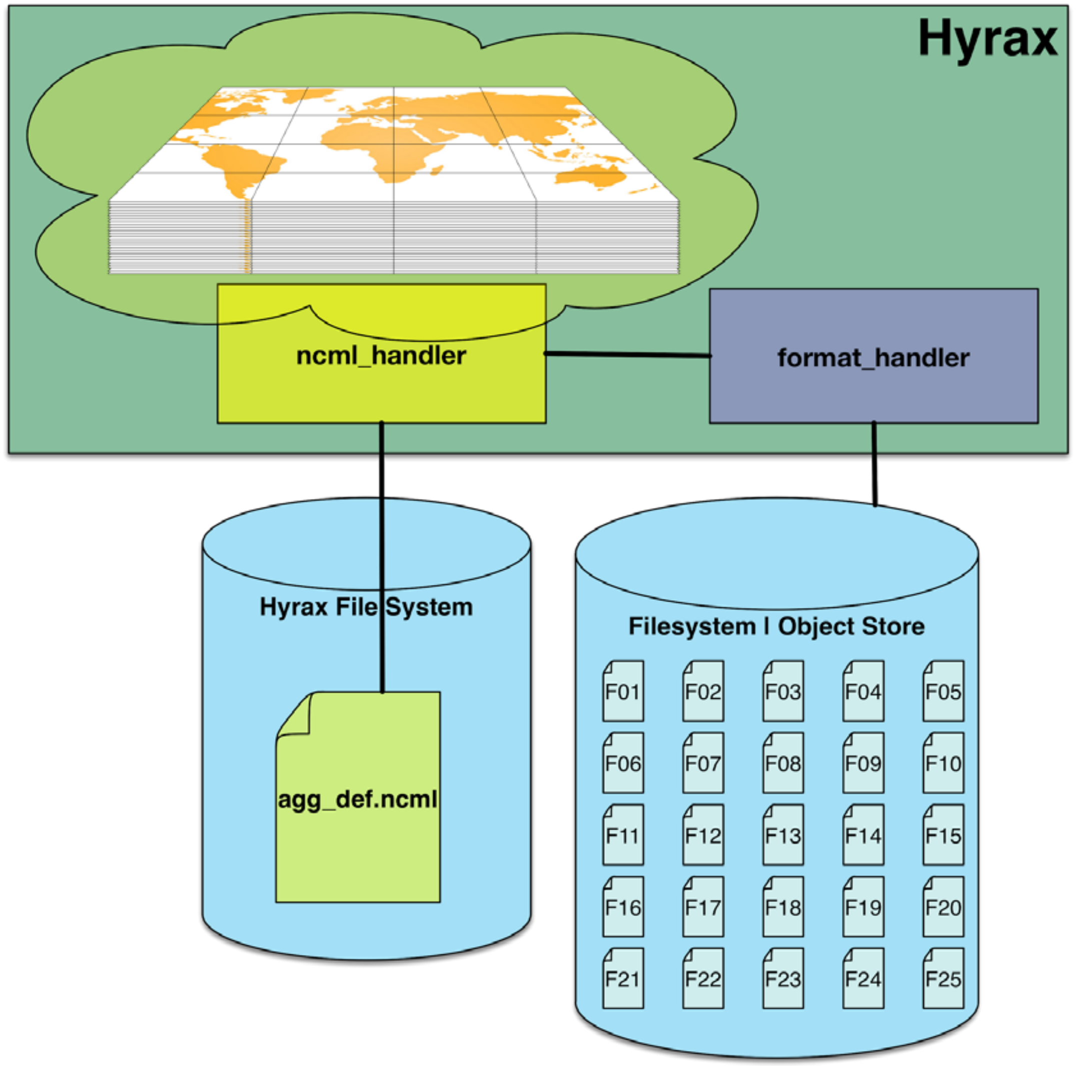
## Aggregations

- Overview – what we mean by 'aggregation'
- Comparison of our current aggregation software (based on NcML)
- How it's possible to mix the old and new software
- And aggregations that use 'virtual sharding' to build higher dimension objects

## What is Aggregation?

- Definition: *noun*, the formation of a number of things into a cluster.
- For OPeNDAP, Aggregations are generally defined using a domain specific language called NcML
- NcML, designed by Unidata, supports several kinds aggregations, including
  - Joining a number of N-dimensional values to form a N+1 dimensional value
  - e.g., combine a series of two-dimensional fields to make a cube
  - Joining a number of N-dimensional values to make a (bigger) N-dimensional value
  - e.g., combine a bunch of cubes to make a new cube

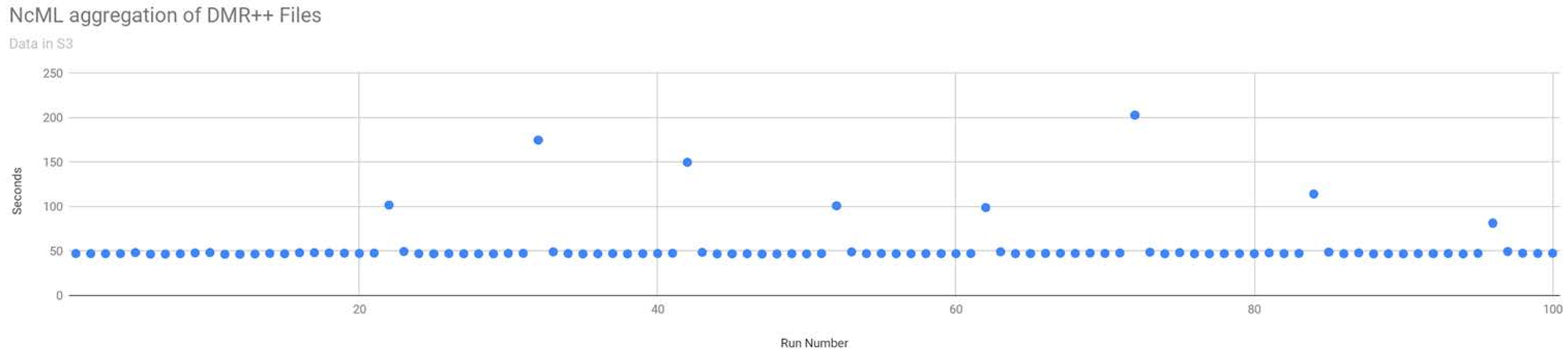
## How NcML Aggregations are Formed by Hyrax



- A NcML file controls which granules are combined
- An interpreter for that file reads it and...
- Uses other parts of the server to read data from those granules
- The result is a virtual data set
- NOTE: The granules can reside on local disk or S3 (File System or Object Store)

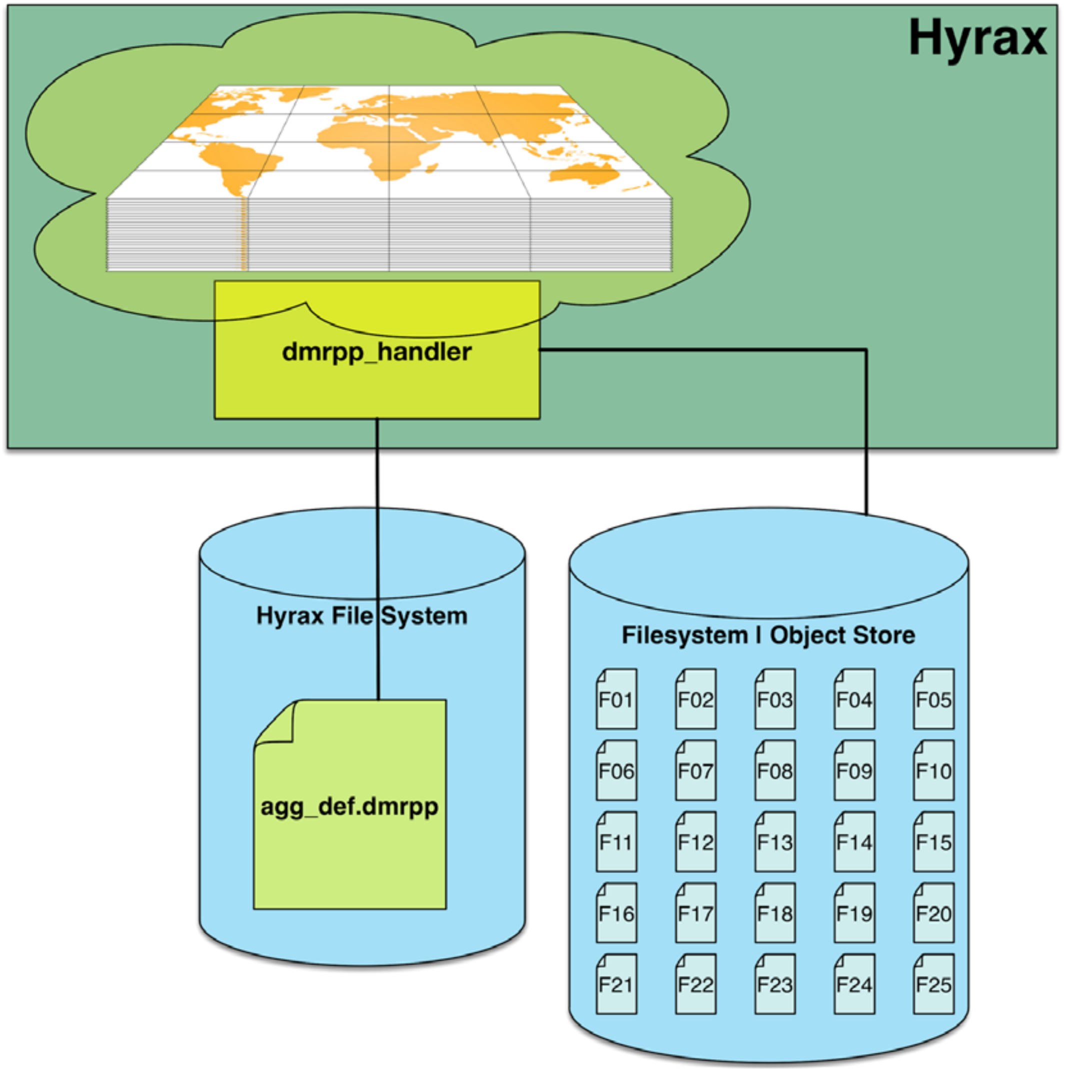
# NcML Aggregations

- NcML aggregation of 365 AIRS[1] files on S3; data accessed using the DMR[2]++ software.
- This show the time taken to access all the data for one variable over all of the 365 days of data
- This is the baseline data for aggregating data stored on S3



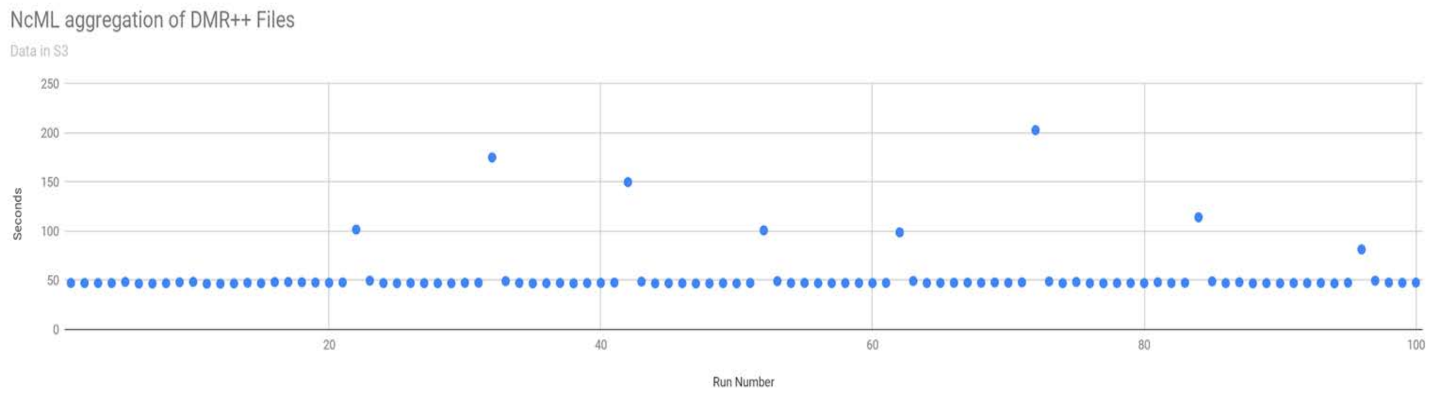
[1]Atmosphereic Infrared Sounder, [2]Dataset Metadata Response

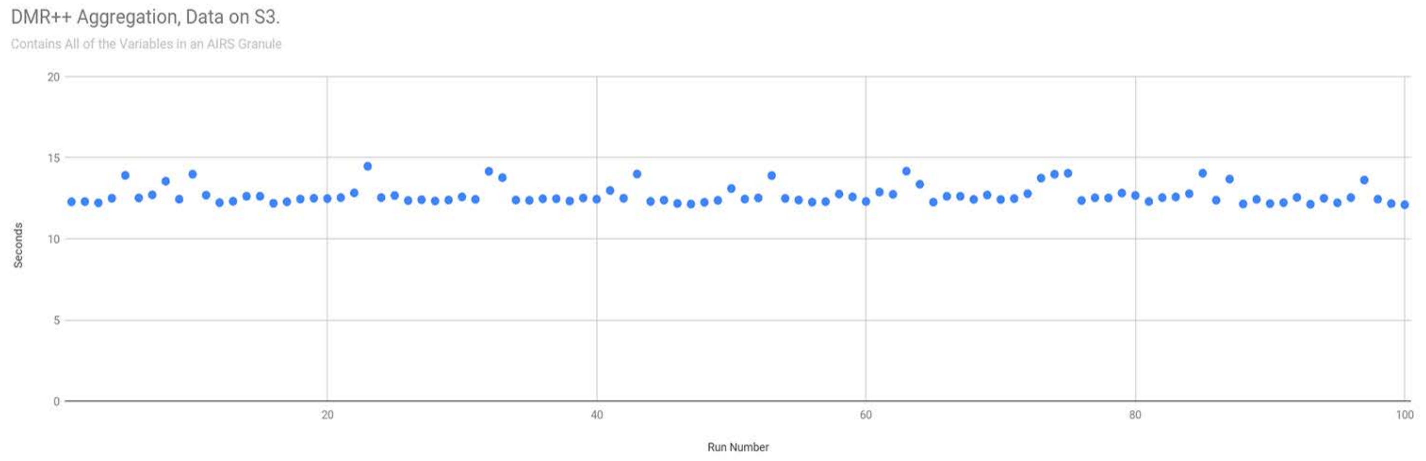
# We can use DMR++ to Define Aggregations



- Instead of writing a NcML file that references a collection of granules,
- ...Write a single DMR++ file that references all of the data.
- This is possible because the 'virtual sharding' technique treats parts of variables as individually addressable 'shards.'

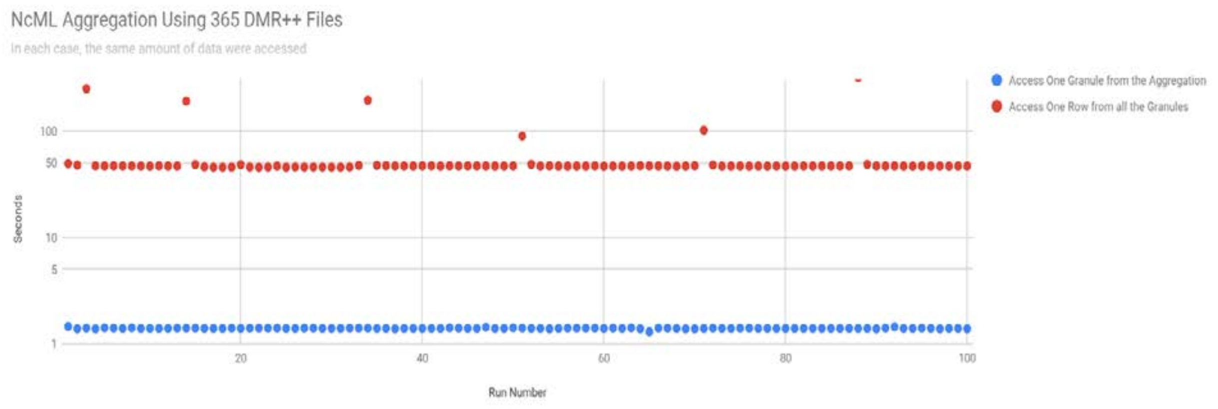
# Comparison of NcML vs DMR++ Aggregations

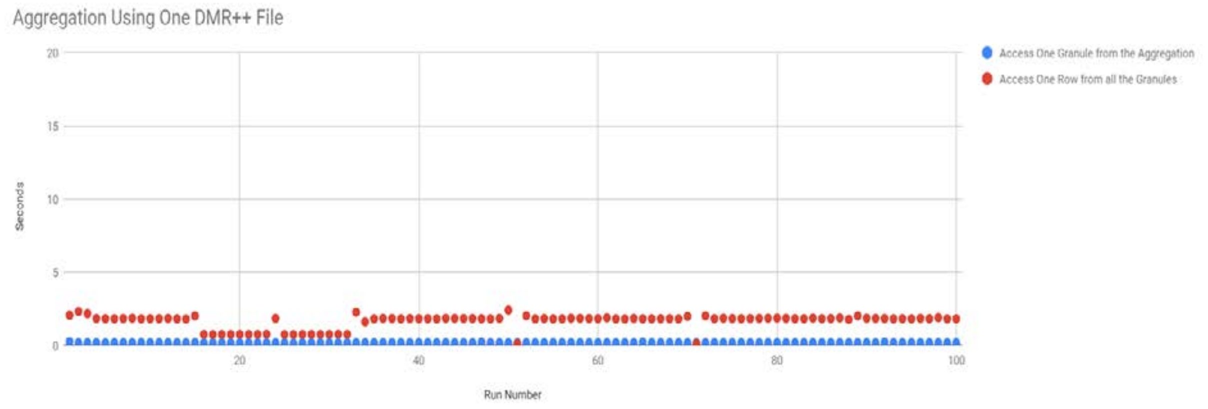




- Difference of ~50s versus ~12.5s
- Access to one variable for all or 365 days
- The aggregation consists of 365 files/objects (one for each day)
- NOTE: These data are for our implementation of NcML

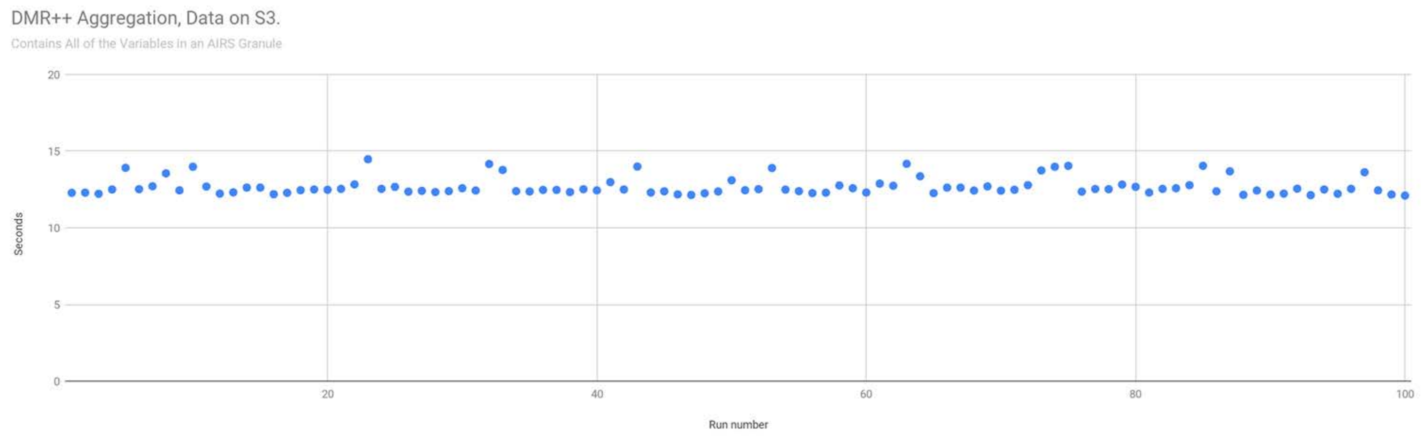
## Orthogonal Accesses – NcML versus DMR++

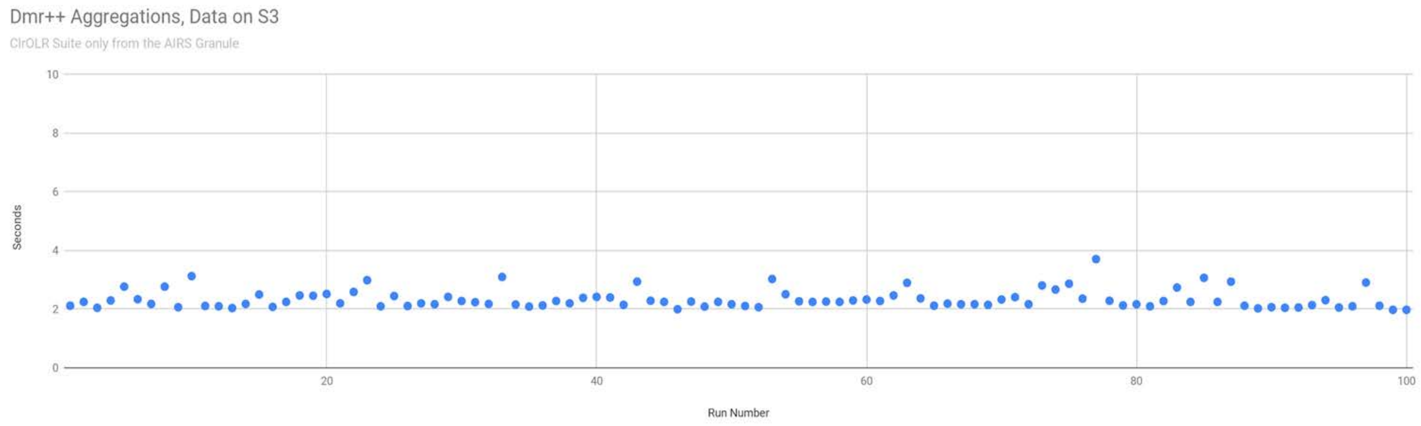




- Slicing across the granules shows the main benefit of this technique
- The 'sharding' aggregation is significantly faster than our implementation of NcML
- Why: Our NcML is processed by an interpreter which iterates over all the needed granule descriptions, while the sharding technique is roughly equivalent to a 'compiled' version of the aggregation

## Processing Large DMR++ XML[1] Files is Expensive





- The same amount of data is returned in each of these two cases
- The DMR++ file in case #2 contains only information for part of the AIRs granule - so it parses much faster
- Optimizing the DMR++ parse and/or caching is worthwhile

[1]eXtensible Markup Language

## Summary

- All five clients work well when reading data from S3 – there is no practical difference in behavior
- NcML aggregations work; we aggregate the 'DMR++' control files
  - Pro: similar to the aggregations built using 'traditional' data files
  - Con: Not as fast, particularly for 'cross-granule' aggregations
- The DMR++ software provides a new way to form aggregations
  - Pro: It can be very fast, with little difference for cross-granule aggregations
  - Con: It is harder to write the aggregation files and results in very large XML documents

This work was supported by NASA/GSFC under Raytheon Co. contract number NNG15HZ39C.

Raytheon

*in partnership with*

ati

Bellfore Consulting Group

Element 84

GENEX SYSTEMS

The HDF Group

INNOVIM The Power of Innovation

MARYLAND HAWK CORPORATION

OPeNDAP

REAN CLOUD

Red Canyon Engineering & Software

STINGER GHAFFARIAN TECHNOLOGIES

VALADOR®

EOSDIS

## Bonus Material – Short version

- How hard will it be to move this code to Google?
- Answer: About 4 hours.
- And, we can cross systems, running Hyrax on AWS or Google and serving the data from S3 or Google GCS.
- Performance was in the same ballpark
- Originally presented at C3DIS, Canberra, May 2019. Four-slide version follows...

## Case Study 2: Web Object Service Interoperability

Moving a System to a different cloud provider

- Given: Hyrax data server running on AWS VMs, and
- Serving data stored in S3
- Move the server to Google Cloud VMs and
- Serve the data from Google Cloud Store

How much modification will the software and data need?

How long will the process take?

Will the two systems have significantly different performance?

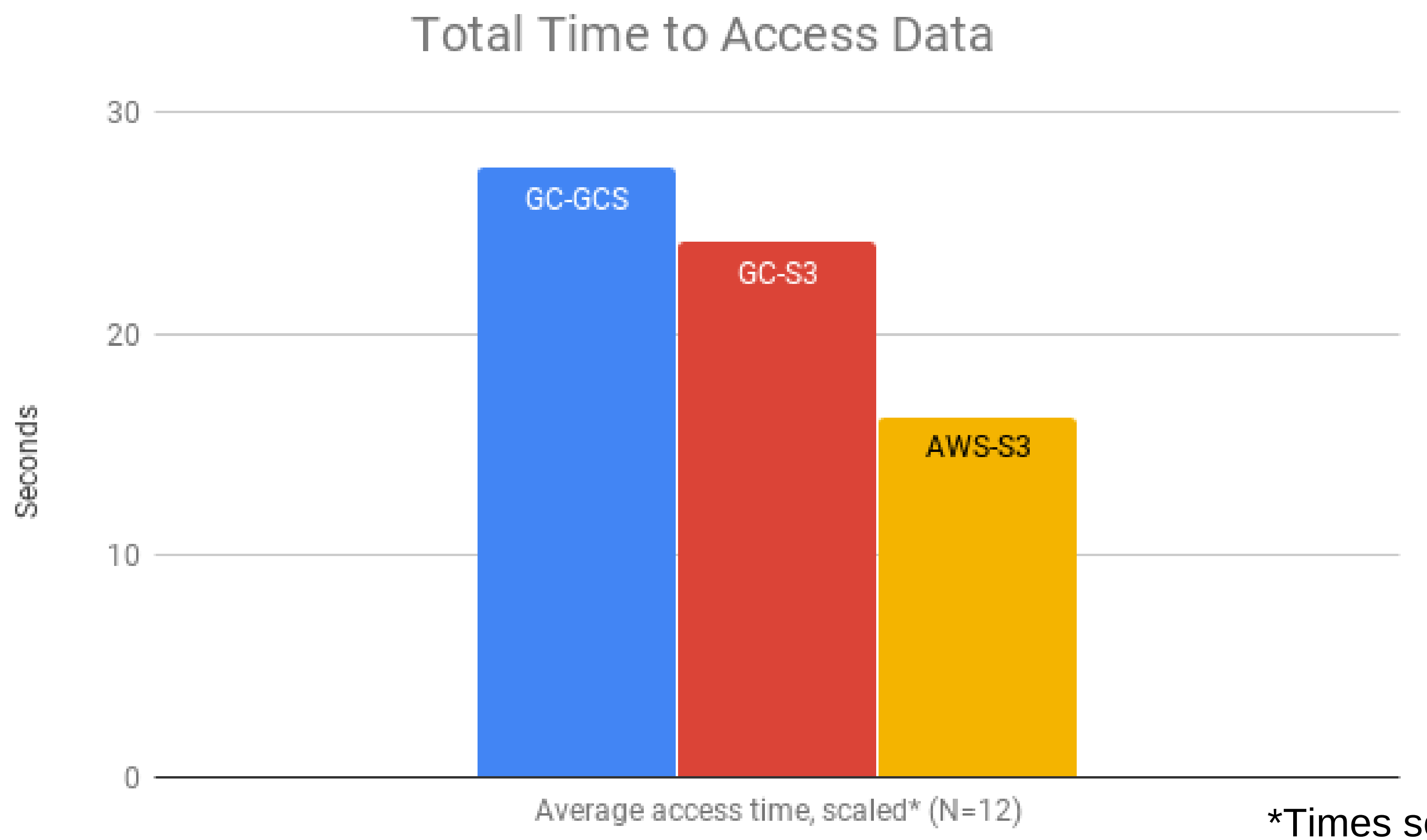
## Case Study Discussion

- The Hyrax server is compiled C++
- The data objects in Amazon S3 were copied to Google GCS
- The metadata describing the data objects were copied and
  - Case 1: were left referencing the data objects in S3
  - Case 2: were modified to reference the copied objects in GCS

No modification to the server software

Time needed to configure the Google cloud systems: *less than 1 day*

## Comparison of Performance



*Times scaled to account for differences in the VM core number

## Case Study 2: Discussion

- Web object store access used the REST API (i.e., the https URLs)
  - Each of the two web object stores behaved 'like a web server'
  - Using common interfaces supports interoperability
  - Other interfaces might not
- Virtual machines
  - I used the same Linux distribution; legacy code known to run there
  - Switching Linux variant would increase the work
- The buckets were public
  - Differences in authentication requirements might require software modification