EOSDIS

NASA'S EARTH OBSERVING SYSTEM DATA AND INFORMATION SYSTEM

Trade Study: Storing NASA HDF5/netCDF-4 Data in the Amazon Cloud and Retrieving Data via Hyrax Server Data Server

Ted Habermann[1], James Gallagher[2], Aleksandar Jelenak[1]
Nathan Potter[2], Joe Lee[1], Kent Yang[1]

[1]The HDF Group [2]OPeNDAP, Inc.

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

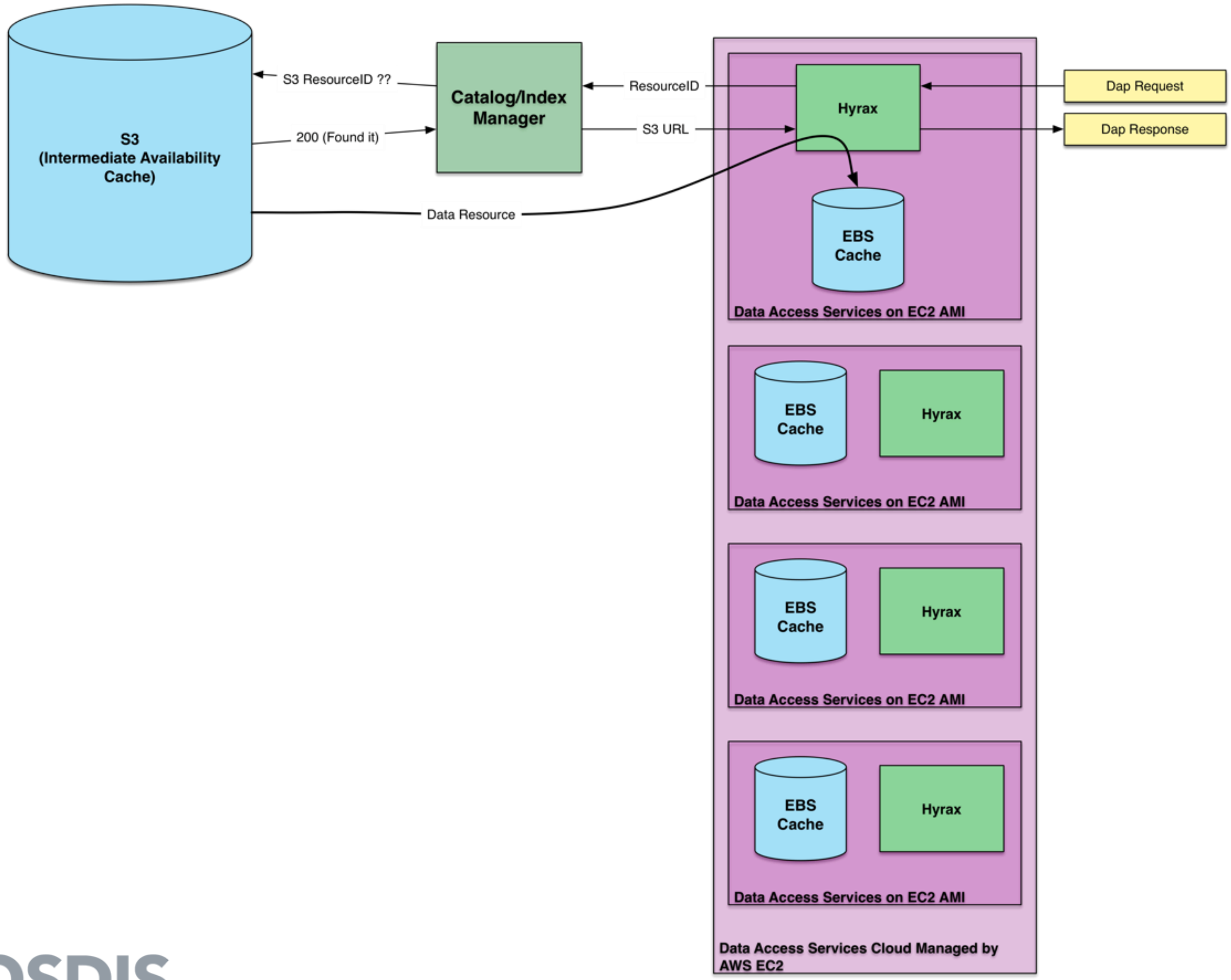
Goals

- Study several approaches to storing and retrieving NASA HDF5 (& netCDF4) data using Amazon Web Services (AWS) Simple Storage Service (S3) and Hyrax server.
- Explore strategies for granulizing and aggregating data that optimize both performance and cost for data storage and retrieval.
- Develop a cloud cost model for the preferred data storage solution that accounts for different granulation and aggregation schemes as well as cost and performance trades.

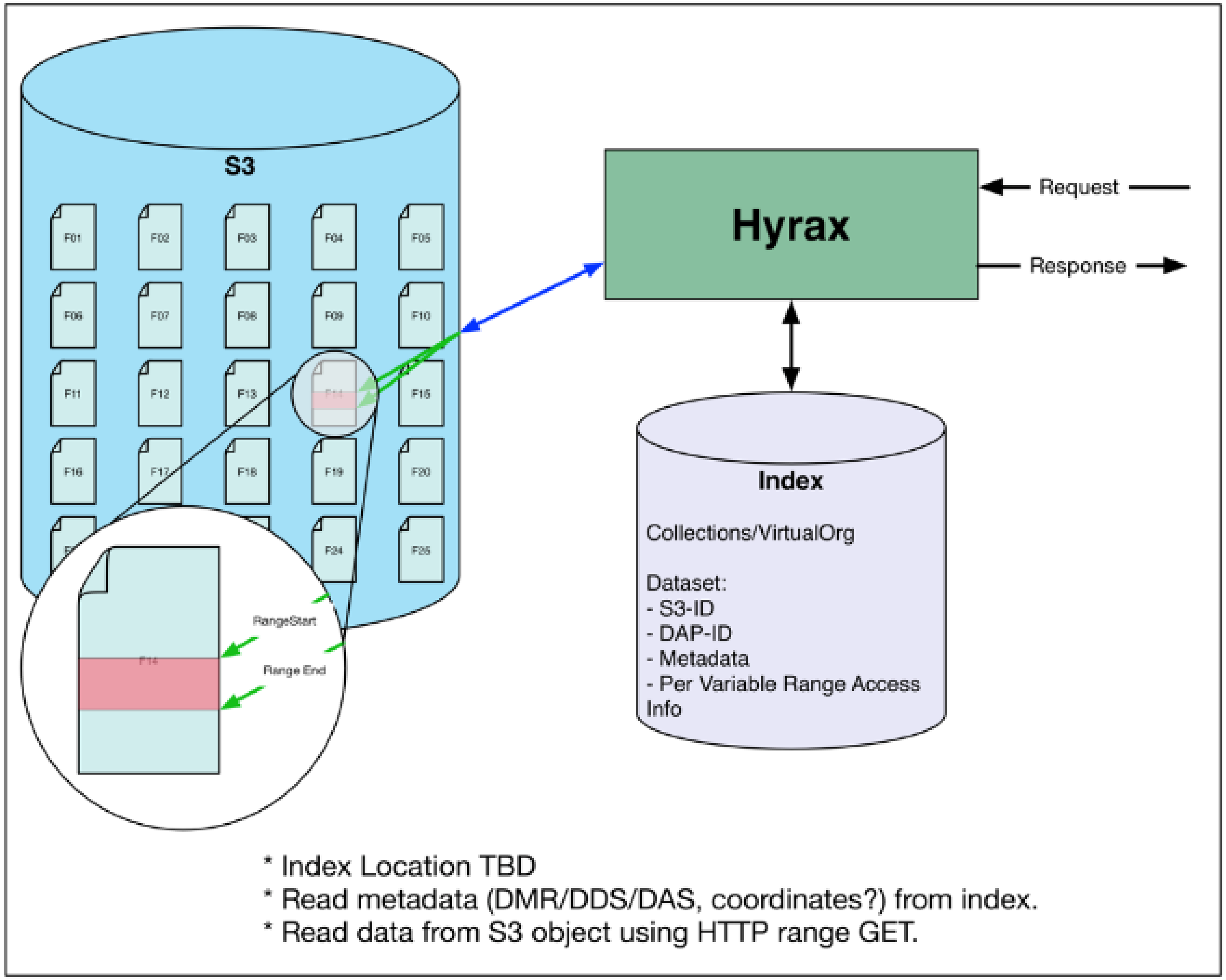
Methodology

- Three architectures explored using proof-of-concept code
- Three sample NASA data collections
- Index files with dataset storage information
- HDF5 library, h5py Python, Hyrax
- Representative use cases with NASA data
- Analysis of performance, access and cost logs

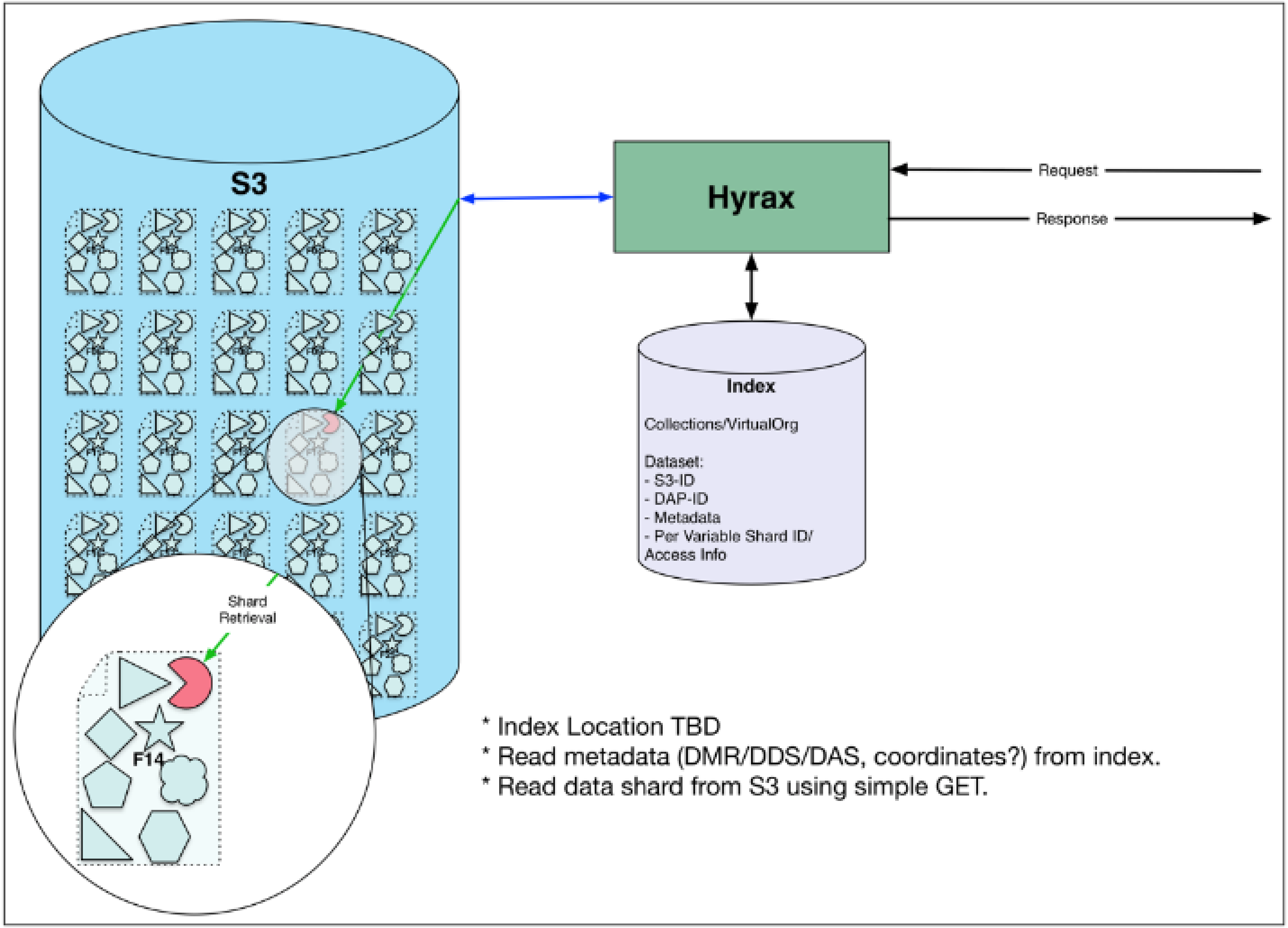
Archit. #1: Baseline Hyrax Data Access



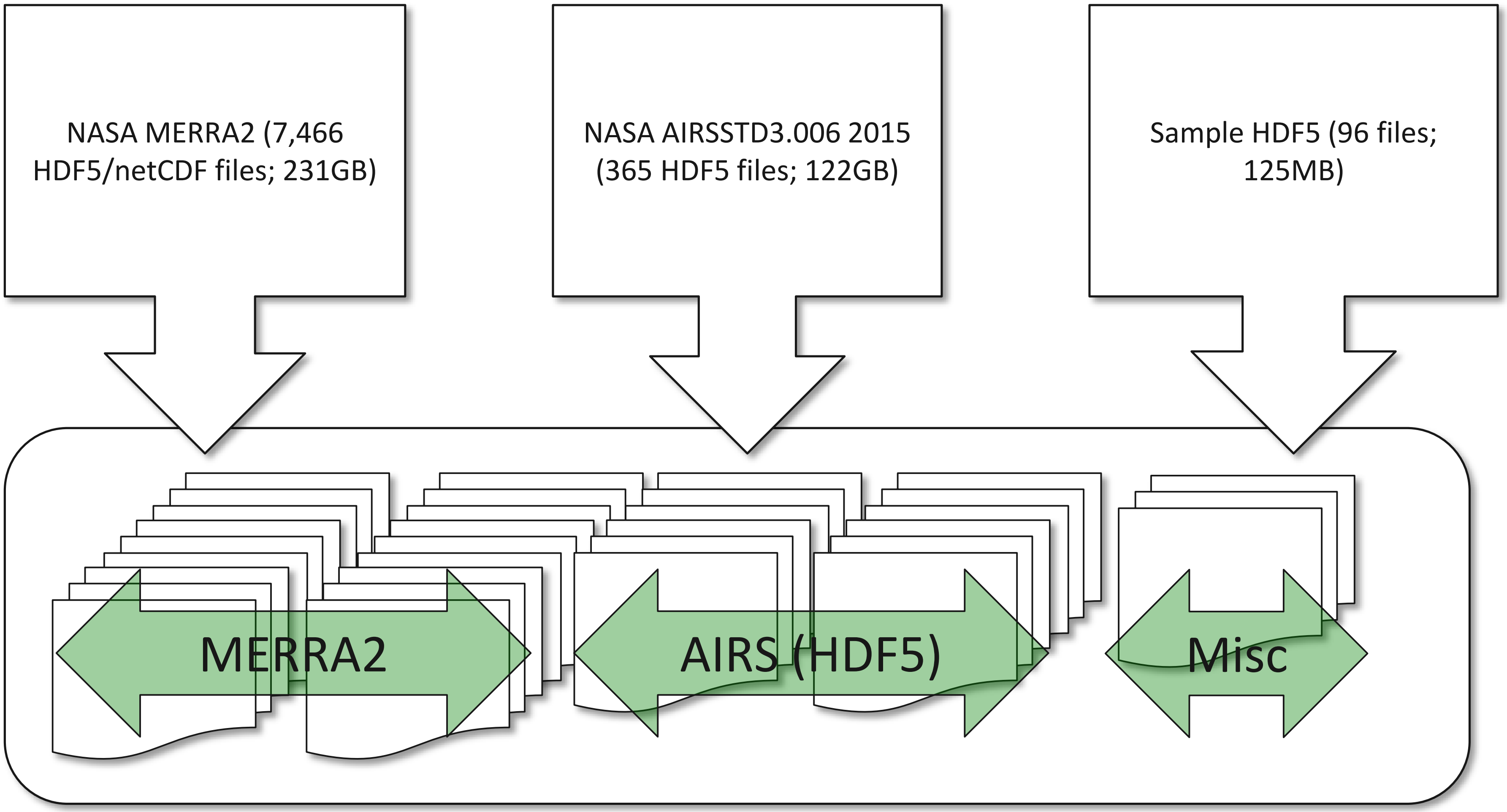
Archit. #2: Files With HTTP Range-Gets



Archit. #3: HDF5 Datasets as S3 Objects



Sample Data Collections in AWS S3



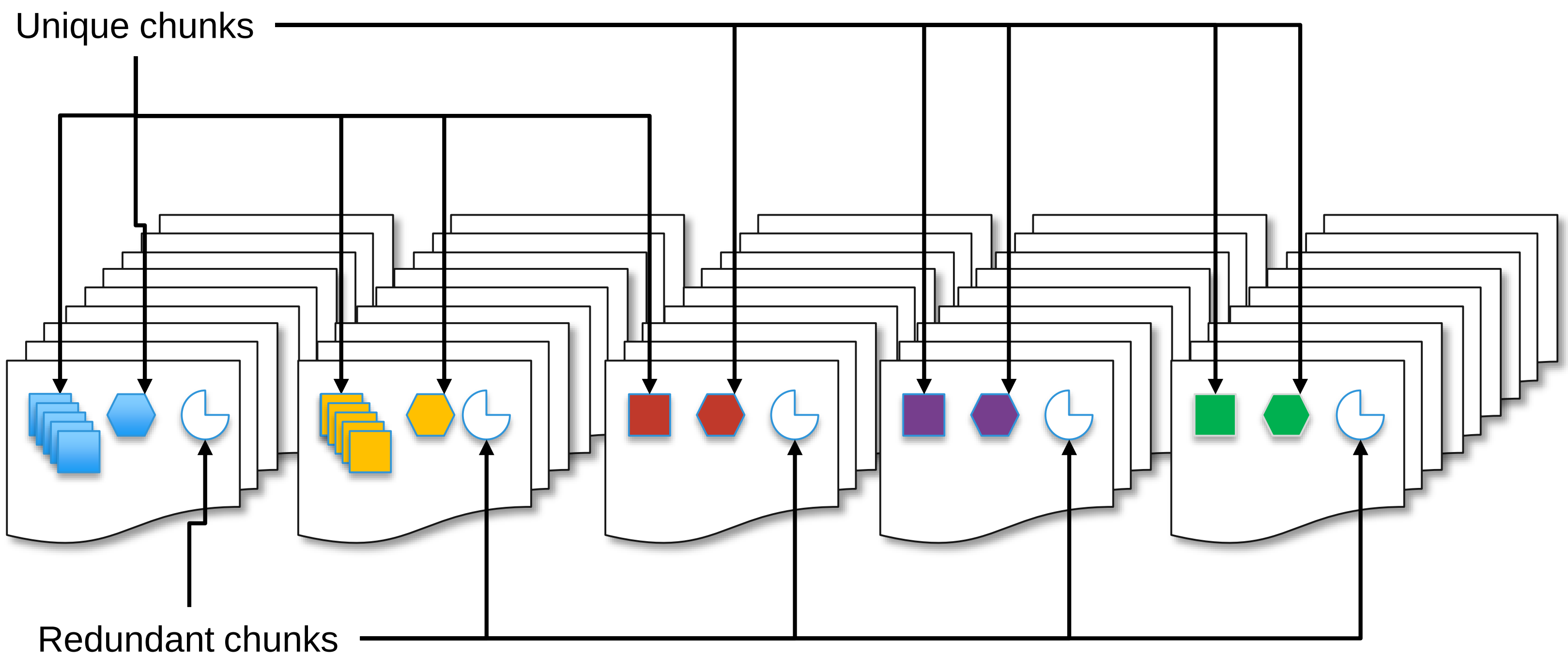
Index Files

- A catalog of file content and dataset byte storage information.
- One for each file in the data collections.
- Hyrax Dataset Metadata Response (DMR) XML used as the ST.
- HDF4 File Map XML used for HDF5 dataset storage information (chunk sizes and offsets).
- Prototyped HDF5 Library to provide this data storage information – tool reads and modifies DMR

Index File Content

```xml
<?xml version='1.0' encoding='UTF-8'?>
<Dataset xmlns="http://xml.opendap.org/ns/DAP/4.0#"
      xmlns:h4="http://www.hdfgroup.org/HDF4/XML/schema/HDF4map/1.0.1"
      dapVersion="4.0" dmrVersion="1.0">
  <Dimension name="Latitude" size="180"/>
  <Dimension name="Longitude" size="360"/>
  <Float32 name="ClrOLR_A">
    <Dim name="/Latitude"/>
    <Dim name="/Longitude"/>
    <h4:chunks deflate_level="2" compressionType="shuffle deflate">
      <h4:chunkDimensionSizes>180 360</h4:chunkDimensionSizes>
      <h4:byteStream nBytes="72049" md5="b707670ae423d0fda9fdb6f33e8f186c"
            chunkPositionInArray="[0,0]" offset="130440821"
            uuid="b0abe13e-4aab-47b3-b256-89d43380600e"/>
    </h4:chunks>
  </Float32>
</Dataset>
```

UUIDs vs. Checksums



Using UUIDs as object identifiers seems like a good choice.

Analysis of the checksums identified identical chunks (same checksum) repeated in every file in a dataset. These chunks can account for a significant portion of the datasets (30-90%).

Storing these chunks once could decrease storage and access costs significantly.

Use Cases

CF responses: Access the CF-enabled DAP4 Hyrax DMR and Data responses.

Default responses: Access the default DAP4 Hyrax DMR and Data responses.

Timeseries request: one pixel MERRA2 files, Query: PRECCU[0:1:*][1][1]

Timeseries request: one pixel AIRS files, Query: Temperature_A[0:1:*][1][1][1]

CF responses (contiguous): Access the CF-enabled DAP4 DMR, Data responses, HDF5 w/ contiguous storage.

2/8 chunks spatial subset: AIRS files, Query: Temperature_A[0:1:*][13:1:15][40:1:45][175:1:195]

Decimated variable: AIRS files, Query: Temperature_A[0:1:*][0:8:23][0:15:179][0:15:359]

4 /16 chunks spatial subset: MERRA2 files, Query: PRECCU[0:1:*][160:1:200][245:1:295]

Decimated variable: MERRA files, Query: PRECCU[0:1:*][0:60:360][0:8:575][0:15:359]

Random spatial subset(10): MERRA2 files, Query: PRECCU[0:1:*][160:1:199][245:1:294]

Random spatial subset(10): AIRS files, Query: Temperature_A[0:1:*][0:8:23][0:1:39][0:1:49]

Random spatial subset(all datasets): MERRA2 files, Query: PRECCU[0:1:*][160:1:199][245:1:294]

Random spatial subset(all datasets): AIRS files, Query: Temperature_A[0:1:*][0:8:23][0:1:39][0:1:49]

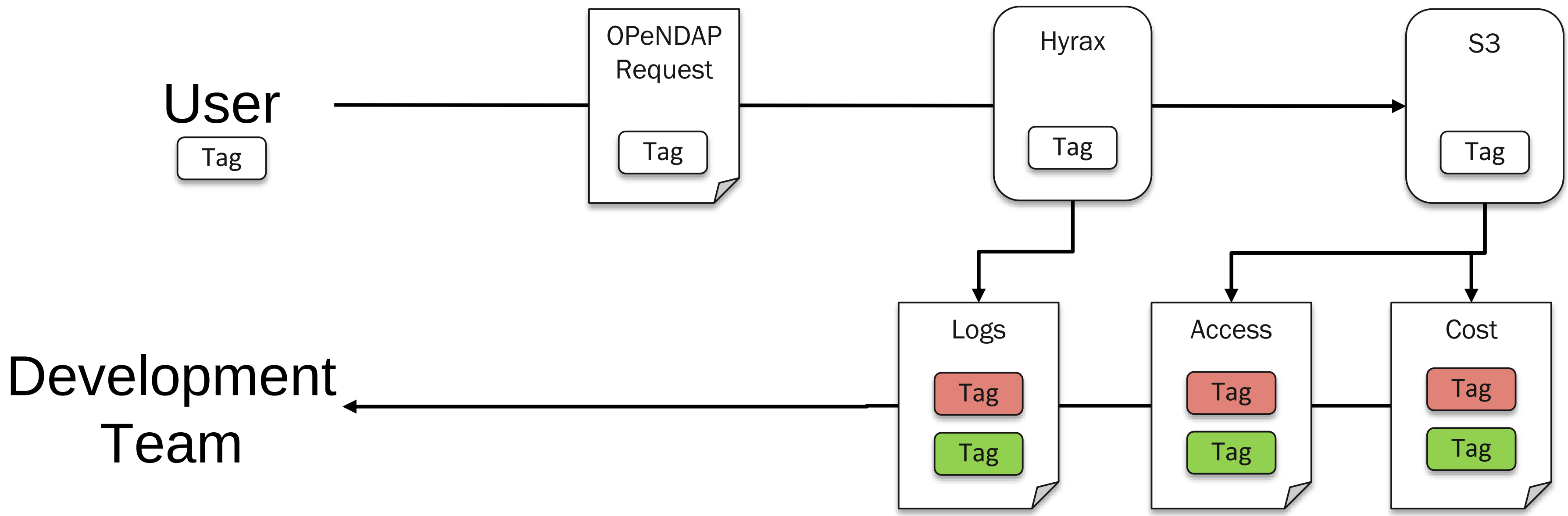
Decimated variable(17): MERRA2 files, Query

Decimated variable(30): AIRS files, Query

All data: 100 MERRA2 files, Query: *none*

All data: 100 of the AIRS files, Query: *none*

Request Tracers

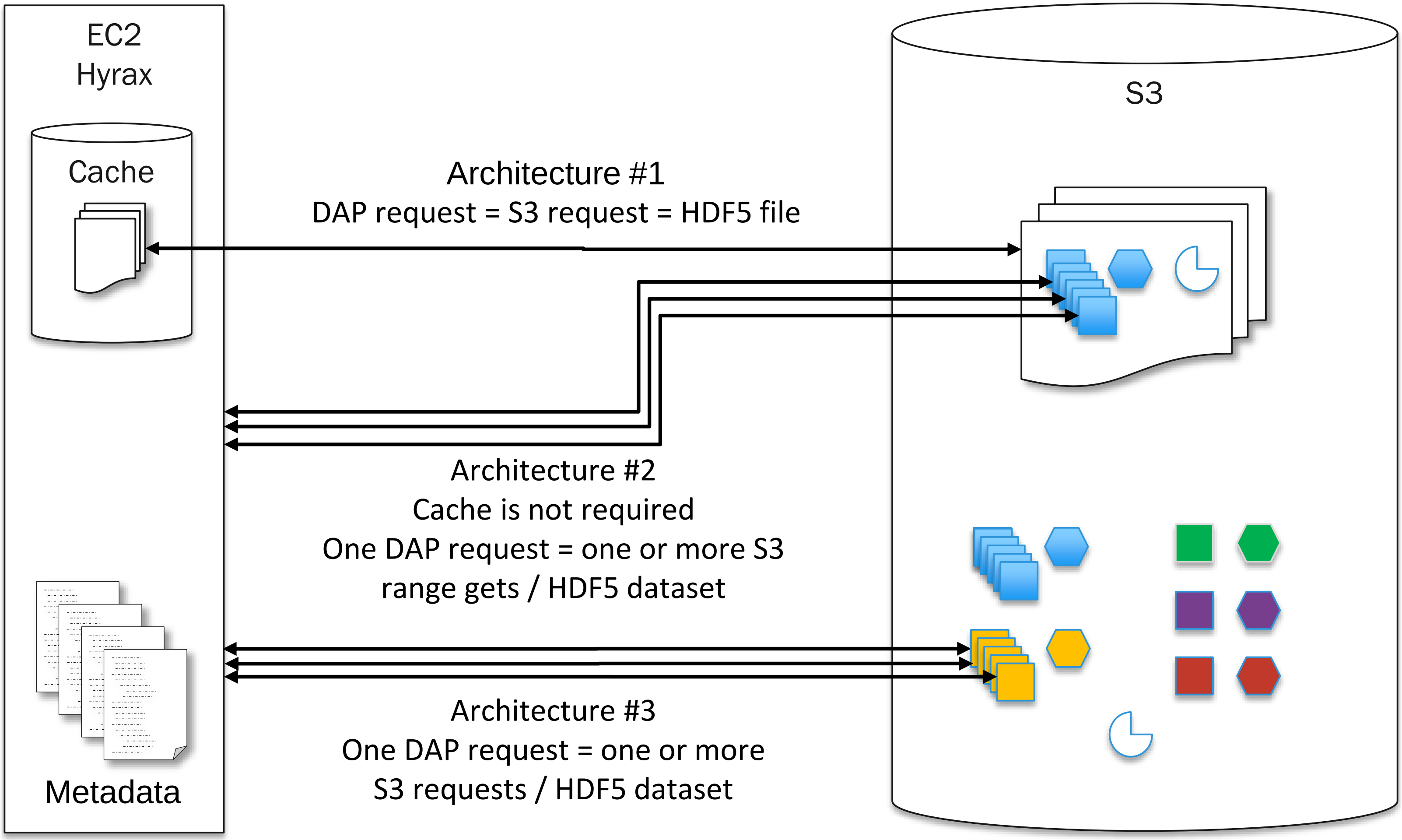


cloudydap={UseCase}_{Arch}_STARTED_{seconds-since-epoch}.h5 where:

- {UseCase} is the use case identifier, e.g. UC1 for the Use Case 1.
- {Arch} is the architecture identifier, e.g. A1CFT for Architecture #1 CF=True
- Hyrax server.*seconds-since-epoch* would be replaced with the output of a date +%s command (ex: 1485208202) which should be the same for every request in a particular run of a collection of the use cases.

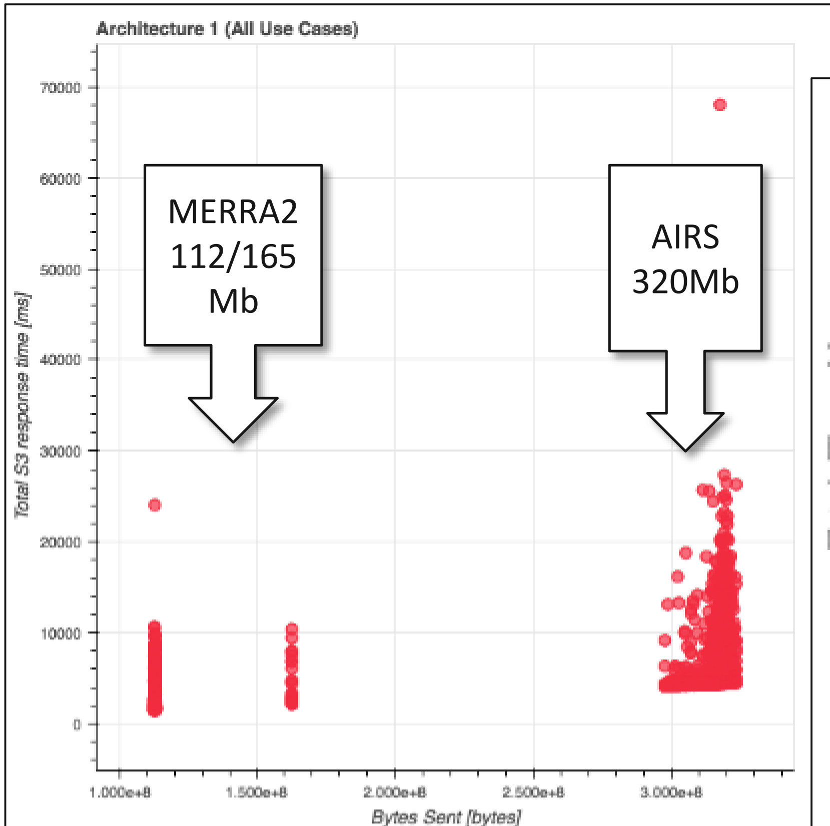
UC1_A1CFT_STARTED_1485208202.h5

Performance / Costs

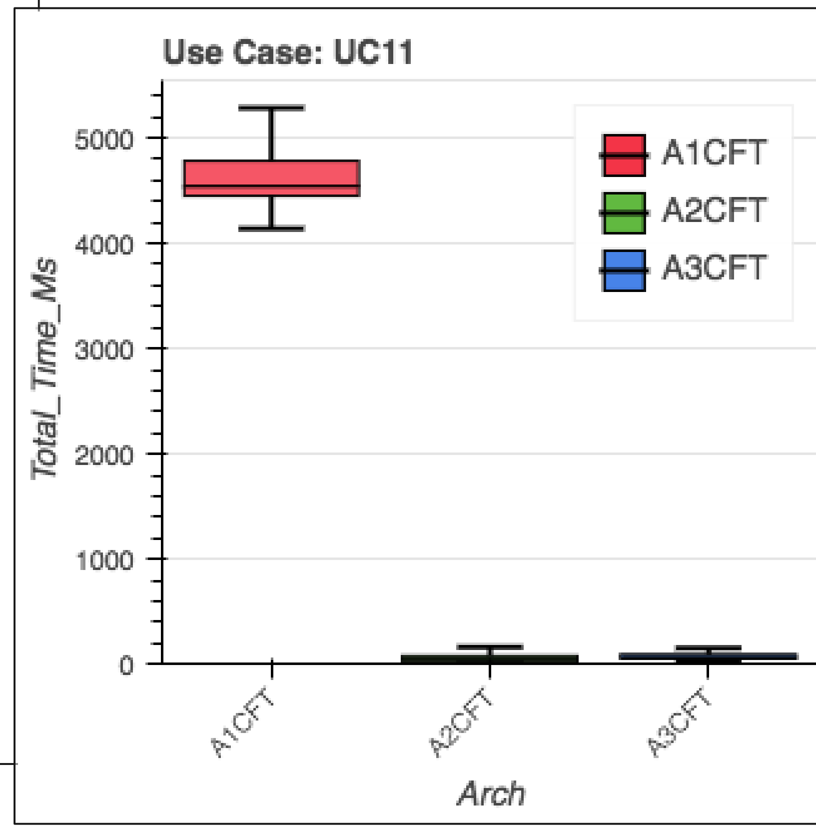


S3 Processing Times

Architecture 1 retrieves entire files – bigger files take longer

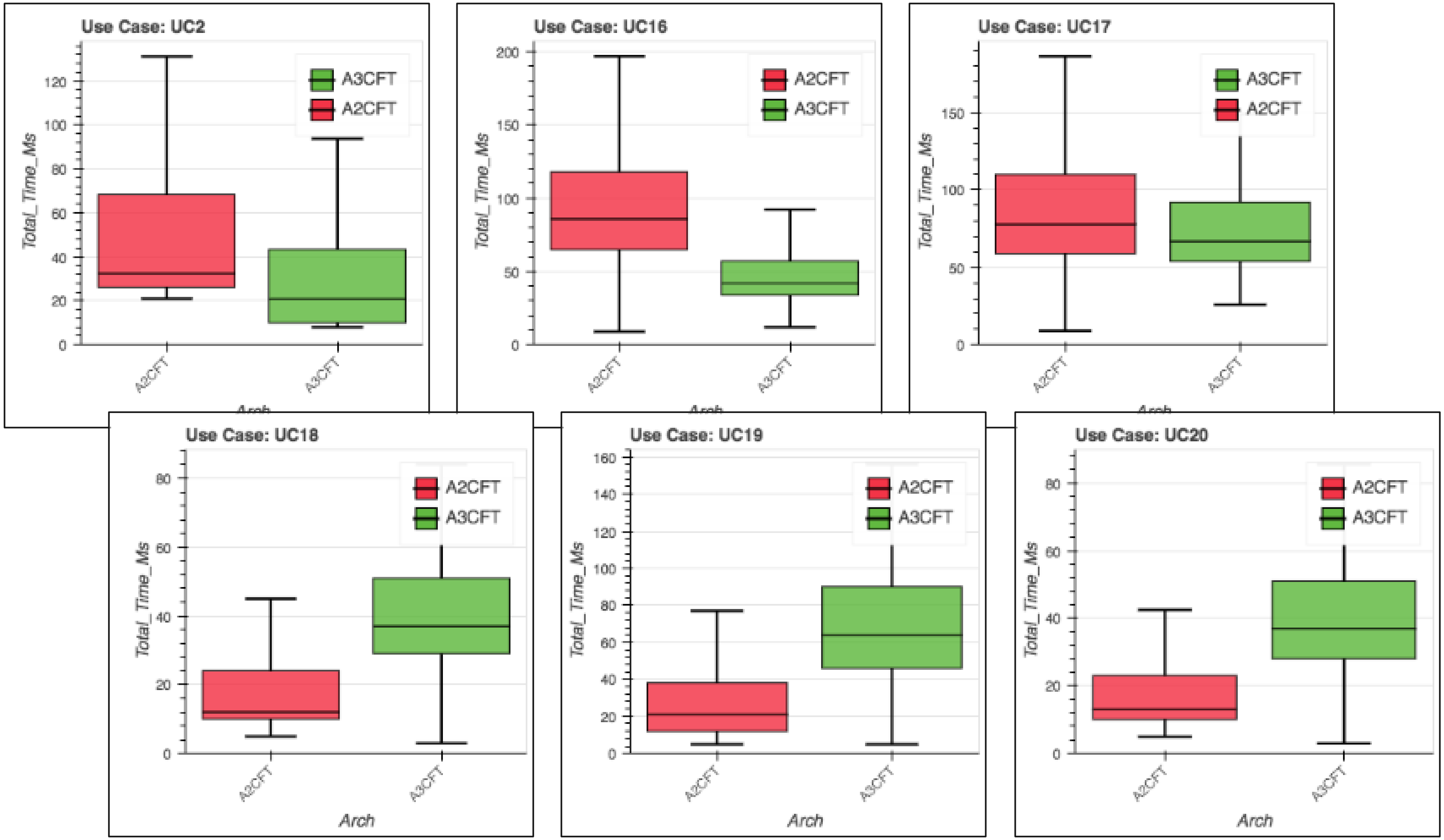


Architecture 2/3 retrieve chunks
small chunks are fast



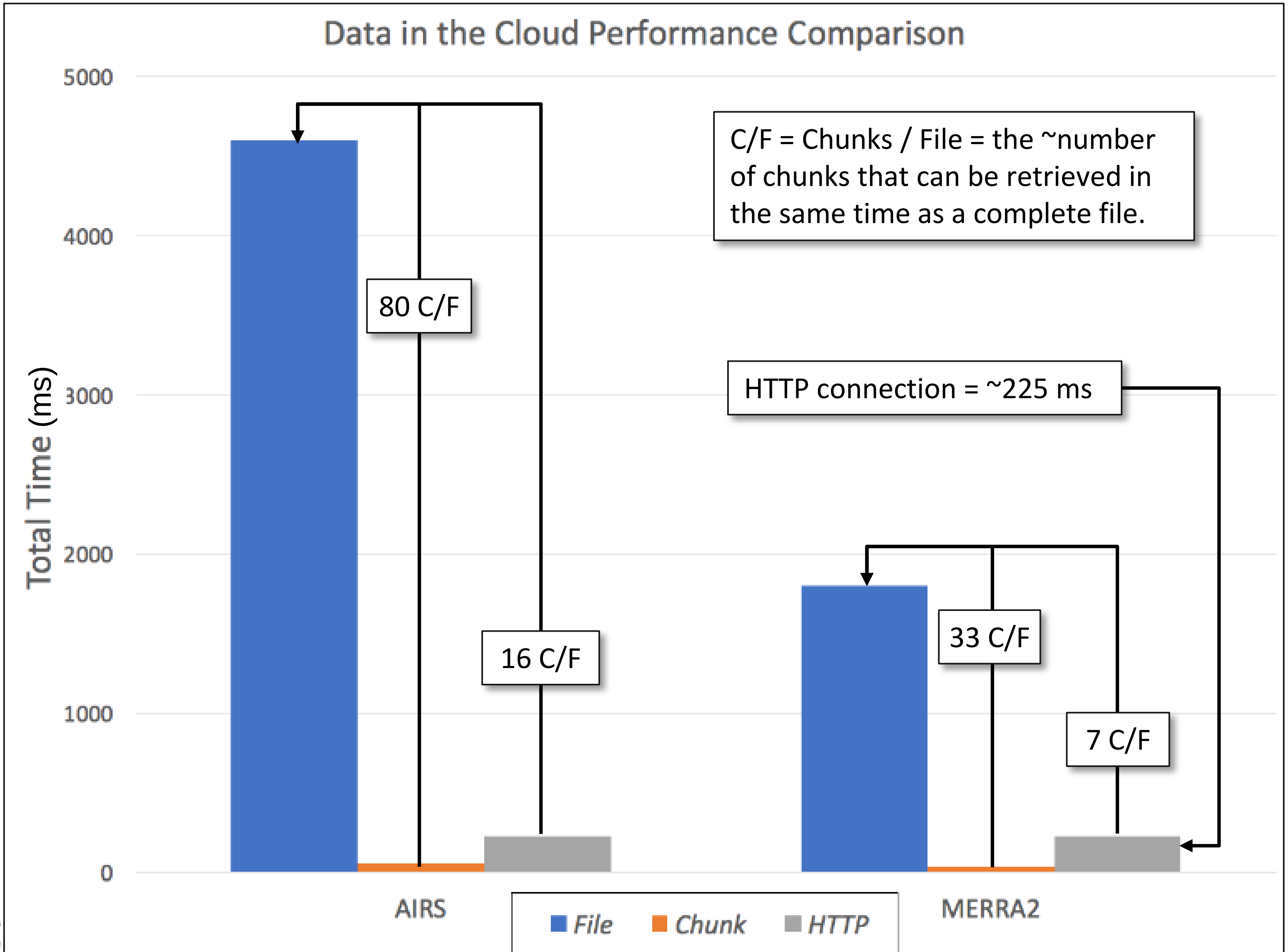
User response times include connection initiation times in addition.

Range-gets vs. Objects



[More Details in Notebook]

Performance Summary



Unrestricted Content

Cost Modelling

- Fixed costs
  - EC2 instances (Hyrax servers) ($$$$)
  - Data / Metadata (?) in S3 ($$$$)
- Dynamic costs
  - Number of Hyrax requests to S3 (¢)
  - Outbound data ($$$)
  - Cache type and size ($$$)
  - Data flow from S3 to Hyrax server(s) ($$ if not in the same AWS region)

Cost Comparison

S3 Storage: $0.022/GB-month

~25% discount??

S3 Request: $0.0000004

EFS cache: $0.3/GB-month
NFS on AWS

EBS cache: $0.1/GB-month
Local disk

One month of EC2 m4.xlarge instance: $156
= 713 GB of storage

Architecture Comparison

	Architecture 1	Architecture 2	Architecture 3
Performance	Faster for requests for large number of variables or entire file Slower for requests for a small number of variables	Faster than A1 for requests accessing small number of variables. Slower for requests for many variables or the entire granule.	
Processing Costs	Depends on processing time		
Storage Costs	~Equal		Can be significantly lower depending on repetition of data values within the granules / dataset.
Original granule retrieval	Yes		No
Data Migration to Cloud	Copy each file to a single S3 object		Shred each file into multiple S3 objects
Commercial Web Crawler Access (GooglеBot, etc.)	Potentially significant if crawlers require large amounts of information to move from S3 to the data server. This situation can be mitigated by limiting crawler access to just metadata held by the server.		

Recommendations

- Integrate support for S3 access into the HDF5 Library
- Model current DAAC data use with detailed / consistent Hyrax logs
- Model and mitigate web crawler costs
- Develop an adaptable server: retrieval strategy depends on nature of request
- Refine the implementations of A1, 2 and 3 (parallel requests, reuse connections, …)
- Explore deployment utilizing a serverless architecture

Acknowledgements

This work was supported by NASA contract number NNG15HZ39C.

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NASA, OPeNDAP Inc., Raytheon or The HDF Group.

OPeNDAP

The HDF Group

Raytheon