Filesystem and object storage for climate data analytics in private clouds with OpenStack

Ezequiel Cimadevilla Álvarez[1],
Aida Palacio Hoz[2], Álvaro López García[2], Antonio S. Cofiño[1]

[1] Santander Meteorology Group, Dep. of Applied Mathematics and Computational Sciences. University of Cantabria, Spain
[2] Advanced Computing Group, Instituto de Física de Cantabria (IFCA - CSIC), Spain

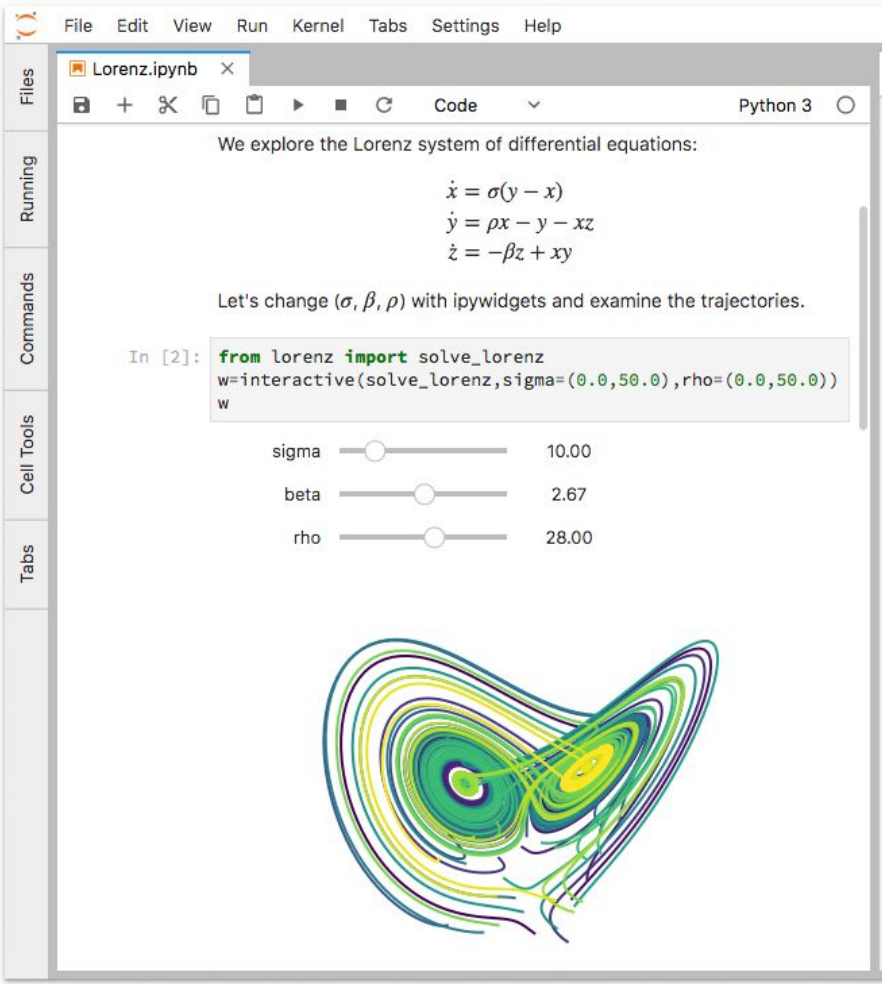
This work has been partially supported by:

- PTA program (2016) from Ministerio de Ciencia e Innovación, Spain
- IS-ENES3 – InfraStructure for the European Network for the Earth System Modelling
- INSIGNIA (CGL2016-79210-R): Contribution to CORDEX Flagship Pilot Studies: regional climate downscaling and data publishing

co-funded by FEDER/ERDF EU funds

Outline

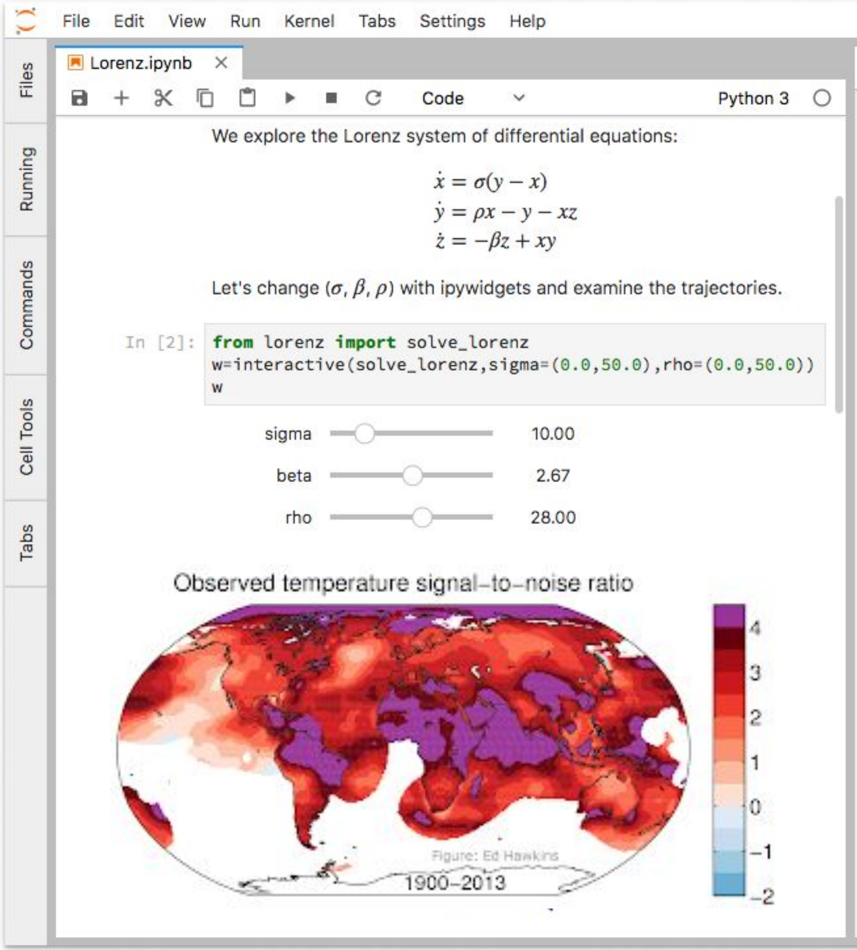
- Data and data analysis in climate science
  - Remote data analysis (Jupyter Notebooks)
- netCDF[1] (really HDF5)
  - Climate data over filesystems
- Chunking
  - How to access large multidimensional data efficiently
- Object storage and clouds
- Zarr
  - Climate data over object storage
- Use case with Openstack and Ceph
- The future of climate data



Data and data analysis in climate science

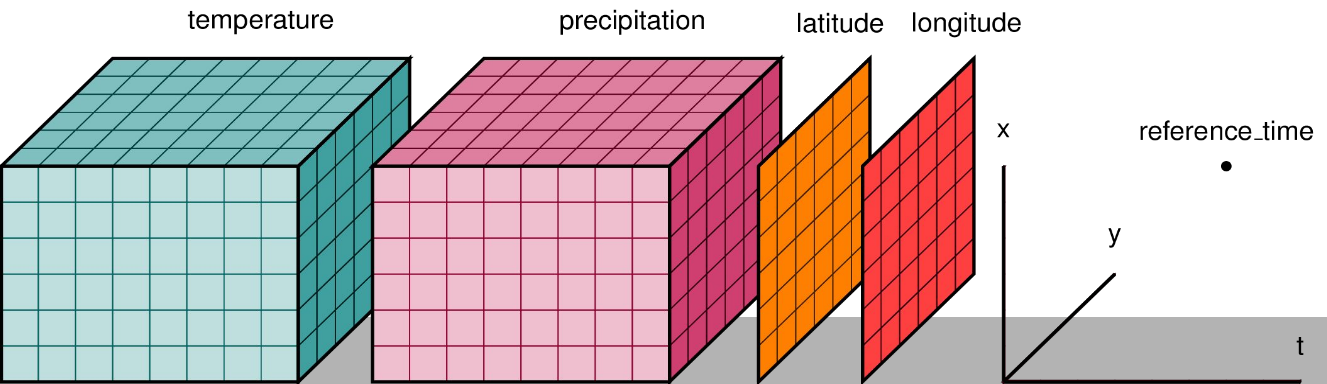
Data analysis in climate science

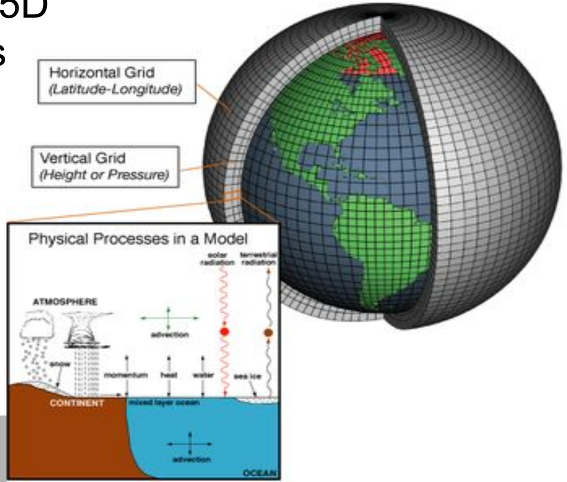
- Climate Change research
  - IPCC Sixth Assessment Report (AR6)
  - Interactive Atlas developed at IFCA (Institute of Physics of Cantabria), Spain
- Heavy increase in size and quantity of data generated by models
  - CMIP3 - 36 TB
  - CMIP5 - 3,3 PB
  - CMIP6 - 100 PB?
- Solution is moving computation (data analysis) near to the data
  - Jupyter Hub Notebooks as interface to remote computation services (**smooth learning curve**)



Data in climate science

- Climate data is the **output of climate models** and/or observations
- Climate data is **multidimensional**
  - X (longitude), Y (latitude), T (time) - 3D
    - precipitation at surface or mean sea level pressure
  - X (longitude), Y (latitude), Z (level), T (time) - 4D
    - temperature at isboraric levels or wind speed profile
  - X (longitude), Y (latitude), Z (level), T (time), E (realization) - 5D
    - multiple model, initializations and parameter ensembles

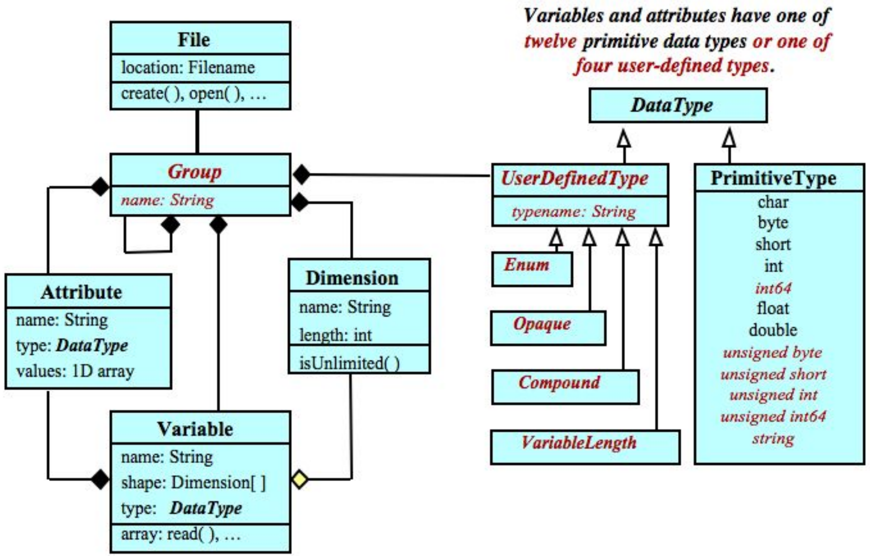




netCDF - The climate community standard

netCDF - Data Model

- "NetCDF (Network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data."
  - Strong commitment for **archival purposes**, libraries with **backward compatibility**
- Developed by Unidata - https://www.unidata.ucar.edu
- Since version 4 (released in 2008), netCDF files are HDF5 files
  - Is possible to implement alternative backends



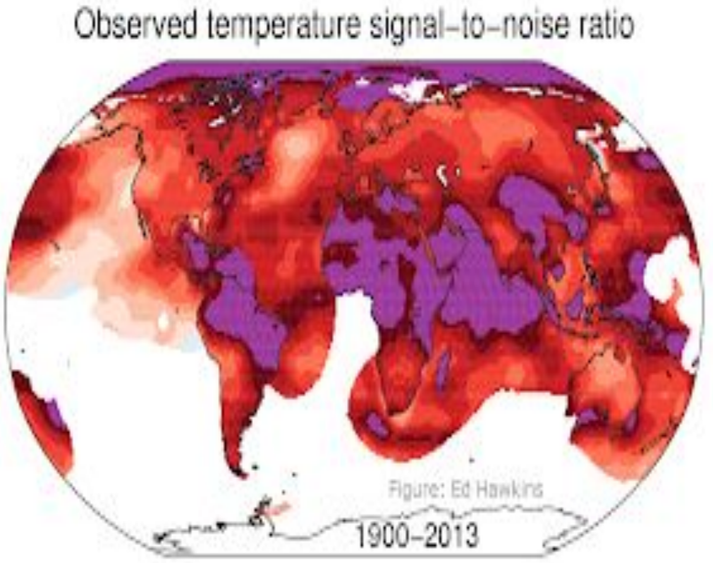
A file has a top-level unnamed group. Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One or more dimensions may be of unlimited length.

netCDF - The climate community standard

- We can work with netCDF files using existing software libraries (e.g. xarray)

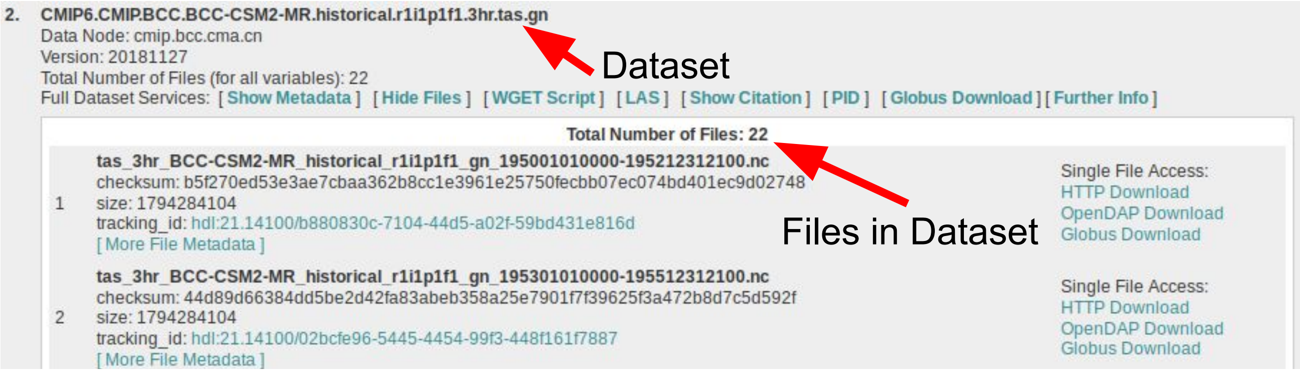
```
In [1]: import xarray as xr
        import numpy as np
        import matplotlib.pyplot as plt
        data = xr.tutorial.load_dataset('air_temperature')
        data

Out[1]: <xarray.Dataset>
        Dimensions:  (lat: 25, lon: 53, time: 2920)
        Coordinates:
          * lat      (lat) float32 75.0 72.5 70.0 67.5 65.0 62.5 60.0 57.5 55.0 52.5 ...
          * lon      (lon) float32 200.0 202.5 205.0 207.5 210.0 212.5 215.0 217.5 ...
          * time     (time) datetime64[ns] 2013-01-01 2013-01-01T06:00:00 ...
        Data variables:
            air      (time, lat, lon) float32 241.2 242.5 243.5 244.0 244.09999 ...
        Attributes:
            Conventions:  COARDS
            title:        4x daily NMC reanalysis (1948)
            description:  Data is from NMC initialized reanalysis\n(4x/day).  These a...
            platform:     Model
            references:   http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanaly...
```



netCDF - The climate community standard

- As **climate models** increase **time** and **spatial resolution**, the **size of files increases**
- To avoid huge file sizes, netCDF files are often **split by time** period **and variable**



- This is convenient for **download and analyze workflow** with only one time period, but it **increases** the number of files to **manage and labor** required to work with the full dataset
- It would be easier to allow applications to subset only specific parts of **remote data**

netCDF - The climate community standard

- **OPeNDAP/DAP - Data Access Protocol**[10]
  - Open access protocol for remote datasets using HTTP since 1993 (a.k.a DODS). DAP4 since 2012
- From python we can use multiple DAP clients to subset remote data using xarray

```
In [14]: url = 'http://193.146.75.233:8080/thredds/dodsC/chunked/tas_AERhr_CNRM-ESM2-1_historical_r1i1p1f2_gr_185001010030-185412312330.nc'
         ds = xarray.open_dataset(url, engine='netcdf4', chunks={'time': 2739, 'lat': 8, 'lon': 32})
```

- Only contents of the dataset that have been requested by client are transferred over the network
  - Only metadata can be retrieved for exploratory purposes
  - Subsetting actions allows to retrieve only portion of the original data
- Client-server architecture which allows different software implementation in both sides
  - THREDDS Data Server based on netcdf-java
  - Xarray, based on netCDF-C library with DAP support

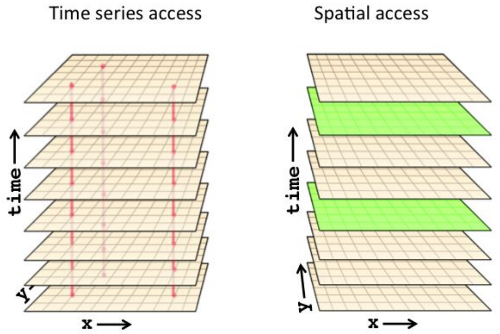
netCDF - Summary

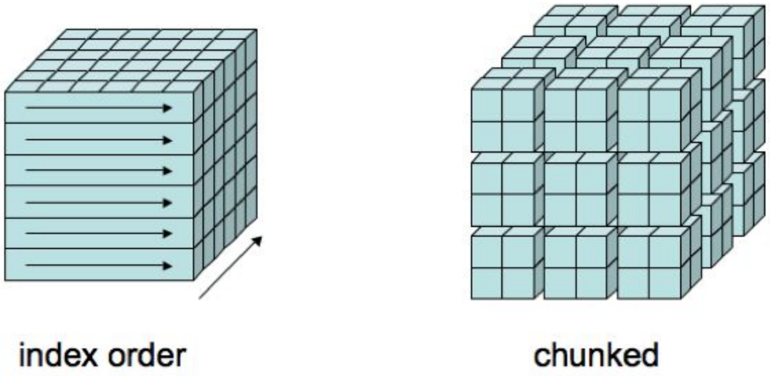
- Climate data - Multidimensional arrays stored in files in the netCDF format
- netCDF-4 files support the HDF5 storage format
- Climate data can be **downloaded** from web repositories (eg. ESGF) to be **locally analysed**
- Climate data can be retrieved remotely through DAP (far more convenient)
  - THREDDS Data Server + netcdf-java on the server side
  - Xarray + netCDF-C on the client side
- Note that we have been talking about **files** that live in POSIX **filesystem** (e.g. ext4, Lustre) to refer to climate datasets
  - These files can be exposed through web services (THREDDS Data Server)
  - Virtual datasets can be composed on the server side (NcML) or on the client side (Xarray)
- How are multidimensional arrays **efficiently** stored under the hood?
  - Answer: Chunks

Chunking - Efficient data access

Chunking - Efficient data access

- Huge tradeoff between different types of access in a **contiguous** stored multidimensional array
- HDF5 files are made of B-trees that store chunks efficiently allowing concurrency, caching and filters





Storage layout, chunk shapes	Read time series (sec)	Read spatial slice (sec)	Performance bias (slowest / fastest)
Contiguous favoring time range	0.013	180.000	14000.0
Contiguous favoring spatial slice	200.000	0.012	17000.0
Default (all axes equal) chunks, 4673 x 12 x 16	1.400	34.000	24.0
36 KB chunks, 92 x 9 x 11	2.400	1.700	1.4
8 KB chunks, 46 x 6 x 8	1.400	1.100	1.2

From: [2]

Object storage and clouds

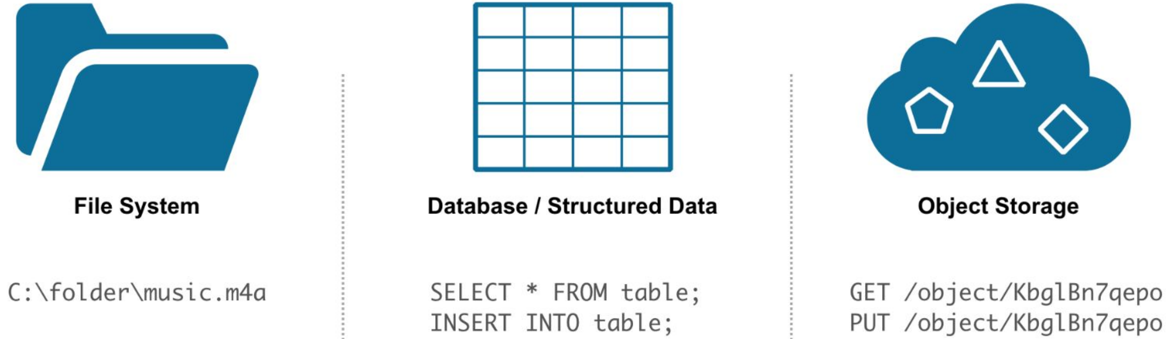
Object storage and clouds

- Commercial cloud providers offer storage in the form of **object storage**

Object storage	**Parallel file systems**
Provided by public cloud vendors (Amazon S3, Google Cloud Storage, ...)	Provided by HPC infrastructures (Lustre, GPFS)
Scalability over consistency	Consistency over scalability
No metadata	File system keeps track of metadata
Atomicity through stateless operations	Stateful operations that requires file system to keep track of operations
No support for partial i/o operations on an object*	Support for partial i/o operations on a file
Flat namespace of key value pairs	Hierarchical namespace of files

Object storage and clouds

- HTTP REST interfaces
- We can take our current netCDF (HDF5) files and drop them into object storage, one key-value pair per file... (note that the usage of slash ‘/’ is part of the key, it is not a hierarchy)
  - PUT /datasets/6hr_precipitation_2015 -> [raw bytes from HDF5 file]
  - GET /datasets/6hr_precipitation_2016 -> [raw bytes from HDF5 file]
  - GET /datasets/6hr_temperature_2015 -> [raw bytes from HDF5 file]
- ...but common client libraries can’t read from object storage, since they only support file systems
- Also, wouldn’t it be more efficient to store metadata and chunks in separate key-value pairs?



Zarr - Multidimensional arrays on object storage

Zarr

- “Python package providing an implementation of chunked, compressed, N-dimensional arrays”[7]
- The storage is no longer a file as in netCDF (HDF5) but everything that implements the MutableMapping interface (key-value) from Python collections
    - File system - The key is the full path of a file and the value is the content of the file
    - Object storage - MutableMapping keys and values match naturally with keys and values in object storage systems
- .zattrs, .zgroup, .zarray are JSON file containing metadata
- Numbered file names are chunks of the multidimensional variable whose name is the directory that contains the chunk files

```
tas_AERhr_CNRM-ESM2-1_historical_r1i1p1f2_gr
├── height
│   ├── 0
│   ├── .zarray
│   └── .zattrs
├── lat
│   ├── 0
│   ├── .zarray
│   └── .zattrs
├── lon
│   ├── 0
│   ├── .zarray
│   └── .zattrs
├── tas
│   ├── 0.0.0
│   ├── 0.0.1
│   ├── 0.0.2
│   ├── 0.0.3
│   ├── 0.0.4
│   ├── 0.0.5
│   ├── 0.0.6
│   ├── 0.0.7
│   ├── 0.1.0
│   ├── 0.1.1
│   ├── .zarray
│   └── .zattrs
├── .zattrs
└── .zgroup
```

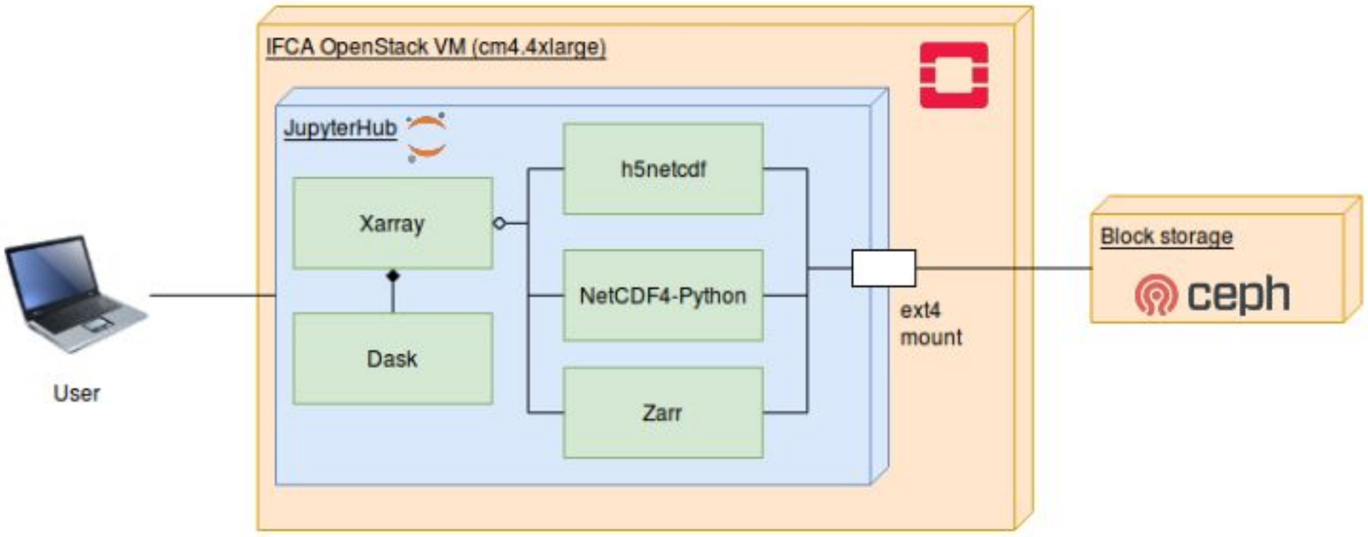
Use case with Openstack and Ceph

Use case with Openstack and Ceph

- The purpose is to illustrate different implementations of a data analysis service deployed in a cloud provider
- Cloud service based on Openstack[9]
- Storage service based on Ceph
  - Ceph uniquely delivers object, block, and file storage in one unified system
  - Ceph RADOSGW is compatible with Amazon S3
- For block storage, one virtual machine includes client applications and data
- For block and object storage access two Openstack virtual machines were created
  - The client virtual machine provides public access to JupyterHub
    - Interactive computing for users through the web browser and Jupyter Notebooks
    - Installed client libraries: xarray, zarr, netCDF4-python, dask
  - The server virtual machine is deployed with the necessary middleware to provide client libraries with access to the data
    - THREDDS Data Server (DAP implementation)

Use case with Openstack and Ceph - Block storage

- Computation and storage on same virtual machine (local access using block storage)
- This scenario was used to measure reference times



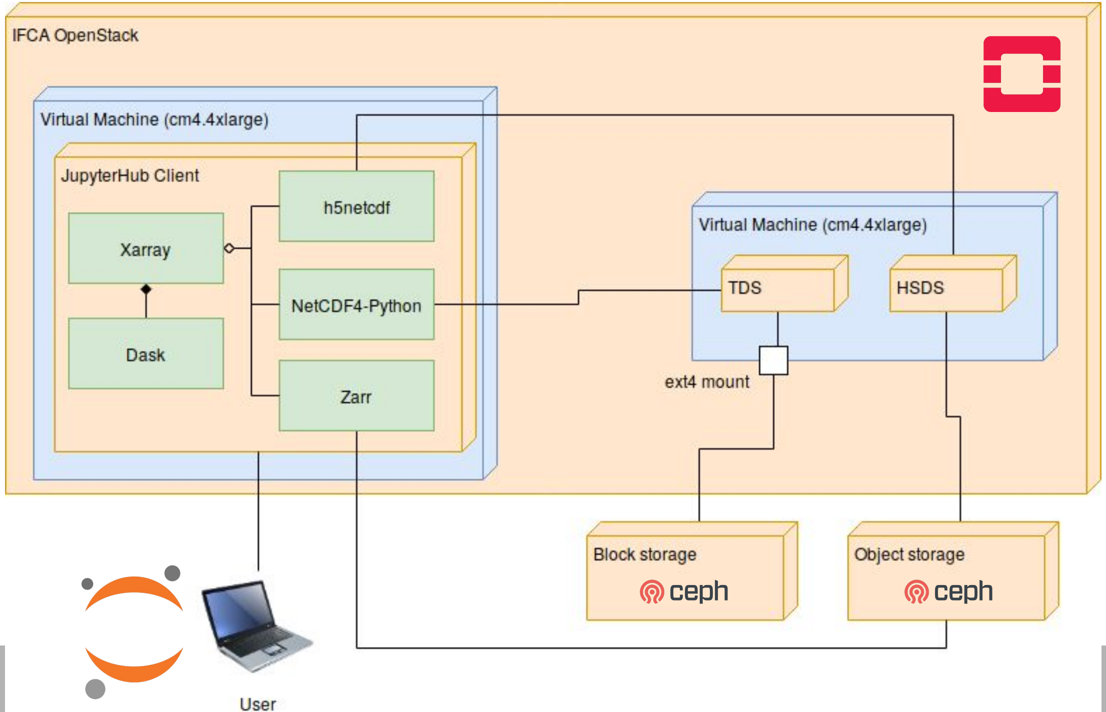
Use case with Openstack and Ceph - Block storage

- Measured times to compute the mean of all temperature values of a 5,4 GB CMIP6 dataset over time axis
- Using xarray + dask for thread parallelism
- For reference, time values were measured using local block storage for both netCDF4-python and zarr

	netCDF4-python	**Zarr**
Serial (no parallelism)	146.2 seconds	148.8 seconds
Dask thread parallelism (16x VCPUs + intel hyperthreading = 32 threads, IFCA cm4.4xlarge[9])	90.5 seconds	49.1 seconds
Speed up (thread over serial)	1.61	3

Use case with Openstack and Ceph - Block and object storage

- Using block storage + TDS for netCDF4-python and object storage for zarr (remote access)



Use case with Openstack and Ceph - Block and object storage

- Using block storage + TDS for netCDF4-python and object storage for zarr (remote access)

	netCDF4-python	Zarr
Serial (no parallelism)	330.6 seconds	477.4 seconds
Dask thread parallelism (16x VCPUs + intel hyperthreading = 32 threads, IFCA cm4.4xlarge[9])	287.9 seconds	60 seconds
Speed up (thread over serial)	1.14	7.95

- Unknown reason for zarr taking too long with parallelism disabled
  - Speed up not meaningful for zarr
- Note degraded speed up for netCDF4-python (1.61 before), also 3x times slower now (90.5 seconds before), how much is middleware (TDS) influencing?
- Threaded zarr time similar compared to block storage (49.1 seconds, 22% slower)
  - No middleware is necessary to access data in object storage

The future of climate data

The future of climate data

- Zarr is gaining traction for use cases that require multidimensional data to be stored in clouds (eg. Pangeo[3])
  - However you don't need zarr to move to the cloud, only if you want to use object storage
- HDF5 (netCDF) is, finding difficulties to adapt to object storage
  - Zarr was added with functionality to read HDF5 datasets stored in object storage[4]
    - It requires to create an additional file that contains the byte position of each chunk inside the file, so it can only work with HTTP Range Requests
  - netCDF-C (4.7.0) can read netCDF files using range requests from S3 endpoints[11]
  - Not general solutions but they are good starting points towards climate data storage in object stores
- The usage of one or another library responds to specific requirements of different use cases
  - e.g. Do I need to modify existing data stored in object storage?
  - Stick to HDF5 (netCDF) whenever possible for climate data analysis, since the broad usage the netCDF library grants compatibility for multiple use cases and archives.

The future of climate data

- The idea of storing multidimensional arrays in chunks already exists in HDF5
- The innovation comes from exposing chunks to client applications instead of keeping them buried inside of a file
  - Exposing chunks to client applications as HTTP resources would remove most of the middleware necessary to remotely subset datasets
    - It would only require a standard HTTP server
  - This is what limits HDF5 to adapt to object storage
  - However HDF5 library is designed of layers that would allow HDF5 datasets to be stored in different ways (Virtual Object Layer and Virtual File Layer)
- The conclusion is that implementing into HDF5 the capability to store chunks in different objects would solve most of the problems
  - Other possible issues: h5py not optimised for multi-threaded access[6] (GIL), compression performance (Blosc)

References

- [1] - Unidata, (2020): NetCDF [software]. Boulder, CO: UCAR/Unidata Program Center. (https://doi.org/10.5065/D6H70CW6)
- [2] - Chunking Data: Why it Matters : Unidata Developer's Blog
- [3] - About Pangeo
- [4] - Cloud-Performant NetCDF4/HDF5 Reading with the Zarr Library
- [5] - To HDF5 and beyond
- [6] - CPU blues
- [7] - Zarr storage specification version 2 — zarr 2.4.0 documentation
- [8] - HDF5 VOL user guide
- [9] - IFCA cloud computing user guide
- [10] - The Data Access Protocol - DAP 2.0
- [11] - NetCDF - Provide byte-range reading of remote datasets

Filesystem and object storage for climate data analytics in private clouds with OpenStack

Ezequiel Cimadevilla Álvarez[1],
Aida Palacio Hoz[2], Álvaro López García[2], Antonio S. Cofiño[1]

[1] Santander Meteorology Group, Dep. of Applied Mathematics and Computational Sciences. University of Cantabria, Spain
[2] Advanced Computing Group, Instituto de Física de Cantabria (IFCA - CSIC), Spain

This work has been partially supported by:
- PTA program (2016) from Ministerio de Ciencia e Innovación, Spain
- IS-ENES3 – InfraStructure for the European Network for the Earth System Modelling
- INSIGNIA (CGL2016-79210-R): Contribution to CORDEX Flagship Pilot Studies: regional climate downscaling and data publishing

co-funded by FEDER/ERDF EU funds