DKRZ

EOSC-Pillar

Current State of our Work

- > 5TB of climate data transferred into DKRZ's swift object storage
    - CMIP6 CMIP (The World Climate Research Programme's Coupled Model Intercomparison Project) & CORDEX WCRP CORDEX
- Developed python package and tutorial notebooks

Why are we using Object Storage?

- Easy to scale and to maintain → cost effective
- Driven by metadata
- Direct HTTP access (no log-in required)

What is Zarr and why are we using it?

- Alternative to NetCDF (Network Common Data Form)
- Provides implementation for chunked N-Dimensional arrays
- Optimized for cloud object storage

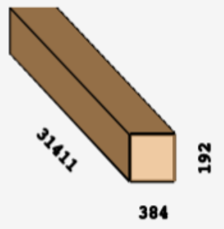
How?

https://gitlab.dkrz.de/mipdata/python_package_zarr_in_swift

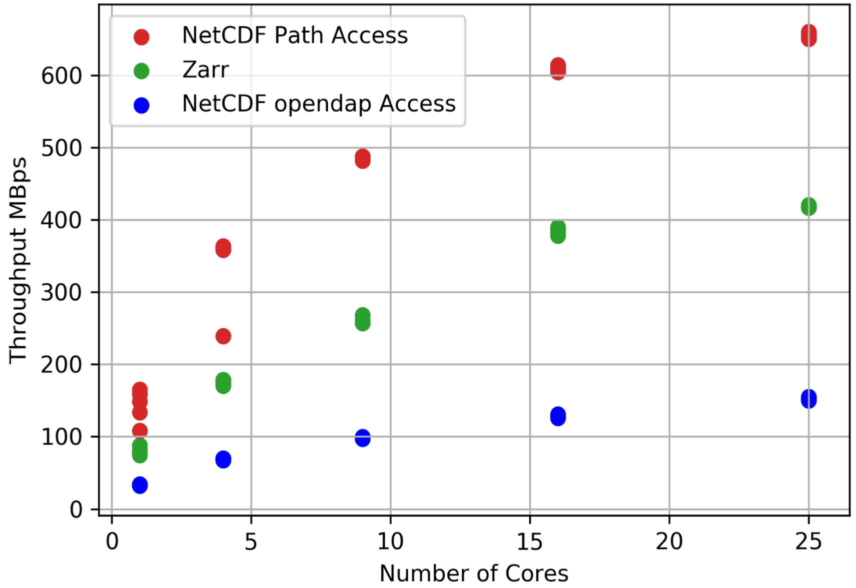
Zarr file opened with Xarray

xarray.DataArray 'tasmax' (time: 31411, lat: 192, lon: 384)

	Array	Chunk
Bytes	9.26 GB	8.85 MB
Shape	(31411, 192, 384)	(30, 192, 384)
Count	1049 Tasks	1048 Chunks
Type	float32	numpy.ndarray



Read-Throughput Comparison



This work has partially been supported by EOSC-Pillar: GA 857650

Transfer Data from NetCDF on Hierarchical Storage to Zarr on Object Storage

CMIP6 Data Use Case

Marco Kulüke
DKRZ
kulueke@dkrz.de

Fabian Wachsmann
DKRZ

Georg Siemund
University of Hamburg

Hannes Thiemann
DKRZ

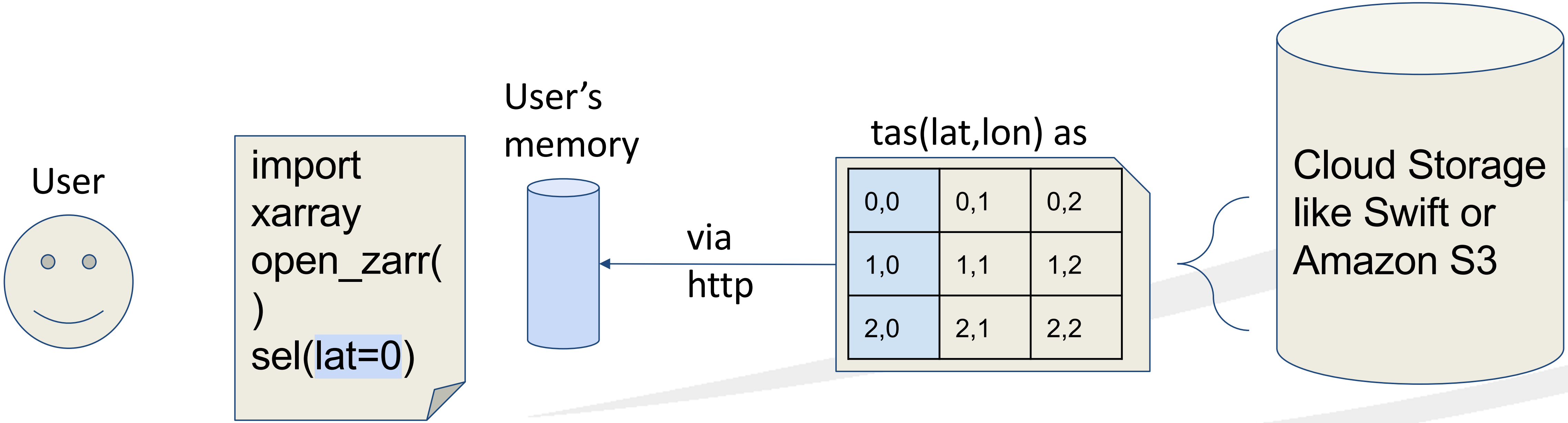
Stephan Kindermann
DKRZ

Abstract

This study provides a guidance to data providers on how to transfer existing NetCDF data from a hierarchical storage system into Zarr to an object storage system.
In recent years, object storage systems became an alternative to traditional hierarchical file systems, because they are easily scalable and offer faster data retrieval, as compared to hierarchical storage systems.
Earth system sciences, and climate science in particular, handle large amounts of data. These data usually are represented as multi-dimensional arrays and traditionally stored in netCDF format on hierarchical file systems. However, the current netCDF-4 format is not yet optimized for object storage systems. NetCDF data transfers from an object storage can only be conducted on file level which results in heavy download volumes. An improvement to mitigate this problem can be the Zarr format, which reduces data transfers, due to the direct chunk and meta data access and hence increases the input/output operation speed in parallel computing environments.
As one of the largest climate data providers worldwide, the German Climate Computing Center (DKRZ) continuously works towards efficient ways to make data accessible for the user. This use case shows the conversion and the transfer of a subset of the Coupled Model Intercomparison Project Phase 6 (CMIP6) climate data archive from netCDF on the hierarchical file system into Zarr to the OpenStack object store, known as Swift, by using the Zarr Python package. Conclusively, this study will evaluate to what extent Zarr formatted climate data on an object storage system is a meaningful addition to the existing high performance computing environment of the DKRZ.

Zarr in Object Storage is promising, because

- User reads directly from the cloud without authentication
- Datasets are “analysis ready”
- Files can be larger than memory → Less I/O, faster processing



Does it keep its promises?

Our questions:

- How performant is Zarr in Swift Object Storage?
- Can Zarr be "conform" to a netCDF standard?

Our long term goals:

- Develop a python library for conversion and archiving in Swift Object Storage

Highly developing topic so we need to keep being updated:

- NetCDF implements a backend support for Zarr

How to work with zarr?

```
In [1]: import xarray as xr
        import intake
        import fsspec
```

- Load required packages
- `fsspec` is a standard interface for opening files

```
In [2]: # Path to catalog descriptor on the DKRZ server
col_url = "https://swift.dkrz.de/v1/dkrz_a44962e3ba914c309a7421573a6949a6/intake-esm/swift-cmip6.json"

# Open the catalog with the intake package and name it "col" as short for "collection"
col = intake.open_esm_datastore(col_url)
```

- Define path to CMIP6 catalog
- Open catalog with `intake`

```
In [3]: # Store the name of the model we chose in a variable named "climate_model"
climate_model = "MPI-ESM1-2-HR" # here we choose Max-Plack Institute's Earth Sytem Model in high resolution

# This is how we tell intake what data we want
query = dict(
    source_id      = climate_model, # the model
    experiment_id  = "ssp370",
    member_id      = "r1i1p1f1",
    variable_id    = "tasmax", # temperature at surface, maximum
    table_id       = "day", # daily maximum
)

# Intake looks for the query we just defined in the catalog of the CMIP6 data pool at DKRZ
cat = col.search(**query)

# Show query results
cat.df
```

Out[3]:

d	source_id	experiment_id	member_id	table_id	variable_id	grid_label	dcpp_init_year	version	time_range	path	zarr_path
Z	MPI-ESM1-2-HR	ssp370	r1i1p1f1	day	tasmax	gn	NaN	v20190710	20150101-20191231	/mnt/lustre02/work/ik1017/CMIP6/data/CMIP6/Sce...	https://swift.dkrz.de/v1/dkrz_a44962e3ba914c30...
Z	MPI-ESM1-2-	ssp370	r1i1p1f1	day	tasmax	gn	NaN	v20190710	20200101-20241231	/mnt/lustre02/work/ik1017/CMIP6	https://swift.dkrz.de

- Define search dictionary (here: maximum daily temperature, ssp370, MPI-ESM)
- Search catalog with `col.search`
- Show results in pandas dataframe
- The last column shows the zarr path

DKRZ

EOSC-Pillar

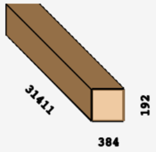
```
In [4]: # Select first zarr path of first file
        selected_path = cat.df["zarr_path"][0]
        selected_path

Out[4]: 'https://swift.dkrz.de/v1/dkrz_a44962e3ba914c309a7421573a6949a6/CMIP6-zarr/ScenarioMIP.DKRZ.MPI-ESM1-2-HR.ssp370.day.
        gn.tasmax/'
```

- Select Zarr path from first entry

```
In [5]: # Define fsmap
        fsmap = fsspec.get_mapper(selected_path)

        # Load Data with the open_zarr) xarray method
        ds_tasmax = xr.open_zarr(fsmap, consolidated=True)

        # Open variable "tasmax" over the whole time range
        tasmax_xr = ds_tasmax["tasmax"]

        tasmax_xr
```

Out[5]: xarray.DataArray 'tasmax' (**time**: 31411, **lat**: 192, **lon**: 384)

	Array	Chunk
Bytes	9.26 GB	8.85 MB
Shape	(31411, 192, 384)	(30, 192, 384)
Count	1049 Tasks	1048 Chunks
Type	float32	numpy.ndarray



▾ Coordinates:

height	()	float64	...
lat	(lat)	float64	-89.28 -88.36 ... 88.36 89.28
lon	(lon)	float64	0.0 0.9375 1.875 ... 358.1 359.1
time	(time)	datetime64[ns]	2015-01-01T12:00:00 ... 2100-12-31T12:00:00

▾ Attributes:

cell_measures :	area: areacella
cell_methods :	area: mean time: maximum
comment :	maximum near-surface (usually, 2 meter) air temperature (add cell_method attribute 'time: max')
history :	2019-07-20T13:41:58Z altered by CMOR: Treated scalar dimension: 'height'. 2019-07-20T13:41:58Z altered by CMOR: replaced missing value flag (-9e+33) with standard missing value (1e+20). 2019-07-20T13:41:58Z altered by CMOR: Converted type from 'd' to 'f'. 2019-07-20T13:41:58Z altered by CMOR: Inverted axis: lat.
long_name :	Daily Maximum Near-Surface Air Temperature
standard_name :	air_temperature
units :	K

- `fsspec.getmapper` creates key-value interface for given URL
- The file will be opened and sliced with `xarray`
- The array has a size of 9,26 GB and is divided into 1048 chunks of 8.85 MB
- The attributes section shows the corresponding metadata

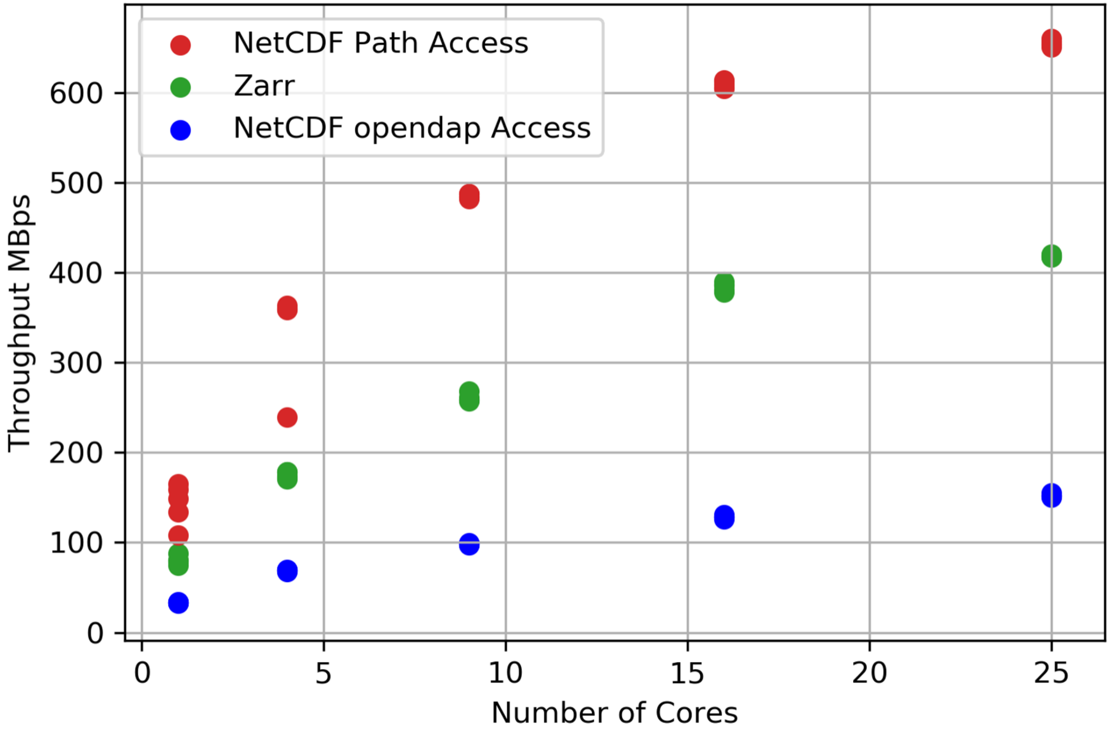
Can Zarr map the CMIP standard?

- We convert 440 different CMIP6 datasets into Zarr and back into netCDF with xarray.
- We compare the original netCDF with the rewritten netCDF.
- Preliminary Result: Zarr is able to map all netCDF features. The data processing software like xarray however need to be adapted to fully copy all original information. We help ourselves with the following reformatting:

```
def format_dset(dset):
    precoords = set(
        ["lat_bnds", "lev_bnds", "ap", "b", "ap_bnds", "b_bnds", "lon_bnds"]
    )
    coords = [x for x in dset.data_vars.variables if x in precoords]
    dset = dset.set_coords(coords)

    dset.encoding["unlimited_dims"] = "time"
    for var in dset.data_vars.variables:
        dset.get(var).encoding["zlib"] = True
        dset.get(var).encoding["complevel"] = 1
```

How does Zarr perform?

- We began to test the zarr read performance with a 10GB Dataset in the Swift Object Storage and compare it with OpenDAP and netCDF on disk.
- We work on a PrePost node on DKRZ's system mistral and use a Jupyterhub-kernel with 256GB memory and 48 cores



GitLab Links

Python Package

https://gitlab.dkrz.de/mipdata/python_package_zarr_in_swift

Tutorials

https://gitlab.dkrz.de/mipdata/zarr-in-swift-objectstorage

Performance Tests

https://gitlab.dkrz.de/b381359/parallelrechnerevaluation-projekt