Interoperable POSIX and Zarr Formats

Charles Zender[1], Charlie Zender[2], Ed Hartnett[2], Dennis Heimbigner[2], and Ward Fisher[2]

[1]Affiliation not available
[2]Departments of Earth System Science and Computer Science, UC Irvine

April 16, 2024

Interoperability Lessons Migrating netCDF Workflows from POSIX to Zarr

Charlie Zender, Departments of Earth System Science and Computer Science, UC Irvine
Ed Hartnett (NOAA), Dennis Heimbigner (Unidata), and Ward Fisher (Unidata)

Introduction

The netCDF Operator (NCO) toolkit introduced support for the object-based Zarr storage format via the NCZarr library in 2022. As of Fall 2023, NCZarr in netCDF 4.9.3-beta supports all commonly used netCDF4 features, including compression and quantization. NCO commands work as expected for all front and back-end storage formats. Operators can ingest and output netCDF3, netCDF4, and/or Zarr backend file formats. The primary interoperability barrier to using workflows built for traditional (POSIX) backend formats on Zarr object stores is filename handling. Zarr object names are URIs incompatible with standard POSIX globbing and wildcards often used to select input files for scripts and Multi-File Operators (MFOs). A new script, **ncz2psx**, combined with standard input/output techniques, emulates globbing and wildcard features used for multi-file operators.

Interoperable POSIX and Zarr Formats

```
in_z="file://${HOME}/in#mode=nczarr,file"
in_p="${HOME}/in.nc"
out_z="file://${HOME}/foo#mode=nczarr,file"
out_p="${HOME}/foo.nc"

ncks ${in_z} # Print Zarr contents
ncks -v var ${in_p} ${out_p} # P->P
ncks -v var ${in_p} ${out_z} # P->Z
ncks -v var ${in_z} ${out_p} # Z->P
ncks -v var ${in_z} ${out_z} # Z->Z
```

NCO Zarr-Compliance and Limitations

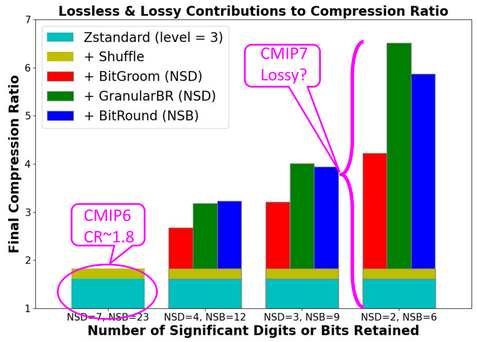
All NCO executables support NCZarr except for **ncatted**, **ncrename** (their functionality for NCZarr is in **ncap2**).

```
ncap2 -s 'RH=0.5' ${in_z} ${out_z}  # Algebra
ncbo ${in1_z} ${in2_z} ${out_z}     # Subtract
ncecat ${in1_z} … ${inN_z} ${out_z} # Ensemble Cat.
nces ${in1_z} … ${inN_z} ${out_z}   # Ensemble Stat.
ncclimo --split ${in1_z} … ${inN_z} # Timeseries
ncflint ${in1_z} ${in2_z} ${out_z}  # Interpolate
ncks --map=map.nc ${in1_z} ${out_z} # Regrid
ncpdq -a lat,lon ${in1_z} ${out_z}  # Permute
ncrcat ${in1_z} … ${inN_z} ${out_z} # Concatenate
ncremap --map=map.nc ${in_z} ${out_z} # Regrid
ncwa -a lat,lon ${in1_z} ${out_z}   # Average
```

Methods

NCZarr is a Zarr-superset adapted to suit the features of the extended (aka netCDF4) Common Data Model (CDM) first supported by the POSIX (HDF5-based) netCDF4 external format. The netCDF library treats datasets in external POSIX and Zarr formats equivalently, with one exception: NCZarr does not yet support user-defined types (enum, vlen, compound). Conversely, Zarr does not support scalars (NCZarr does). To support NCZarr in NCO, we had only to update NCO's file management routines to support NCZarr object trees in addition to "normal" POSIX files.

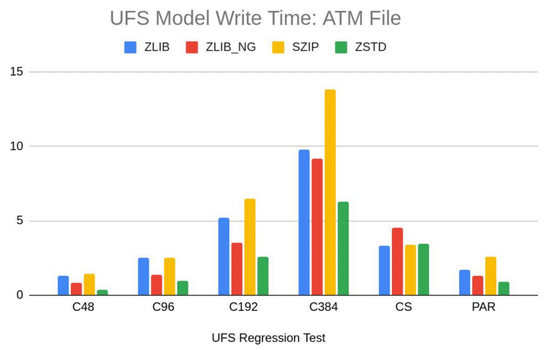
NCO leverages netCDF library access to system codecs (Blosc, Bzip2, Zstd) and transparently encodes/decodes and/or quantizes datasets as necessary. Fully back-compatible, supports MPI with Parallel I/O (PIO) library in C/Fortran.



Quantization Produces Compressible Bit Patterns

Granular BitRound π = 3.14159265 to 1 ≤ *NSD* ≤ 8 significant digits

Sign	Exponent	Fraction (significand)	Decimal	Notes
0	10000000	10010010000111111011011	$3.14159265\overline{}$	Exact
0	10000000	10010010000111111011011	$3.1415926\overline{8}$	NSD = 8
0	10000000	10010010000111111011100	$3.1415929\overline{8}$	NSD = 7
0	10000000	10010010000111111100000	$3.141592\overline{9}393$	NSD = 6
0	10000000	10010010001000000000000	$3.14\overline{1}60156$	NSD = 5
0	10000000	10010010001000000000000	$3.14\overline{1}60156$	NSD = 4
0	10000000	10010010000000000000000	$3.1\overline{4}062500$	NSD = 3
0	10000000	10010000000000000000000	$3.\overline{1}2500000$	NSD = 2
0	10000000	10000000000000000000000	$\overline{3}.00000000$	NSD = 1



ncz2psx "Globs" Zarr Datasets for Multi-File Operators (MFOs)

NCO's new script ncz2psx prepends a desired Zarr scheme (e.g., "**file://**") and appends a fragment (e.g., "**#mode=nczarr,file**") to names. This reduces tedious typing for MFO input:

```
ncra in*_p.nc out_p.nc              # Glob list of POSIX input files
ls in*_p.nc | ncra out_p.nc         # Pipe globbed list to stdin
$ ls -d in1 | ncz2psx               # "file://in1#mode=nczarr,file"
ls -d in1 | ncz2psx | ncremap ${out_z} # Zarr single input via stdin
ls -d in* | ncz2psx | ncra ${out_z}   # Zarr MFO input via stdin
ls -d in* | ncz2psx | ncrcat ${out_z} # Zarr MFO input via stdin
ls -d in* | ncz2psx --scheme=file --mode=nczarr,file | ncra ${out_z}
```

Interoperable, Customizable Compression and Quantization

```
ncks --cmp='gbr|shf|zst' ${in_p} ${out_z} # Quant/Cmp POSIX
ncks --cmp='gbr|shf|zst' ${in_z} ${out_z} # Quant/Cmp Zarr
ncra --cmp='gbr|shf|zst' ${in_z} ${out_z} # Quant/Cmp Zarr
ncremap --cmp='gbr|shf|zst' ${in_z} ${out_z} # Quant/Cmp Zarr
ncks --cmp='zst' ...      # Invoke Zstandard (no Shuffle)
ncks --cmp='shf|zst' ... # Invoke Shuffle then Zstandard
ncks --cmp='shf|bls' ... # Blosc (not HDF5) Shuffle, Blosc LZ
ncks --ppc dfl=3 --cmp='shf|zst' ... # Reasonable default
ncks --ppc dfl=3#Q.?=5#FS.?,FL.?=4 --cmp='shf|zst' … # Custom
ncks --baa=4 --ppc dfl=3#Q.?=5#FS.?,FL.?=4 --cmp='shf|zst' ...
ncks --baa=8 --ppc dflt=9#Q.?=15#FS.?,FL.?=12 --cmp='shf|zst' ...
ls -d in* | ncz2psx | ncra --cmp='gbr|shf|zst' ${out_z}
ls -d in* | ncz2psx | ncrcat --cmp='gbr|shf|zst' ${out_z}
```

Why (Lossily) Compress?

- Datacenters use ~1-3% of global electrical power
- Storage accounts for ~40% of datacenter **emissions**
- Scientific computing archives mostly **false precision**
- Same storage cost for higher resolution simulations

Our improvements to netCDF reduce the power requirements and thus greenhouse gas emissions during long term storage of floating point scientific data. Research workflows that employ these methods are more **sustainable, economical**, and **ethical**.

Takeaways

netCDF treats extended (aka netCDF4) Common Data Model datasets equivalently whether their backend storage format is POSIX (HDF5), Zarr, or other (e.g., DAP2/DAP4, PnetCDF). This enables NCO to manipulate (print, subset, hyperslab, annotate, regrid, perform arithmetic) Zarr datasets with familiar commands. Migration from POSIX to Zarr is straightforward, except for handling dataset name changes from POSIX paths to Zarr URIs in some multi-file workflows. Translator scripts (here, ncz2psx) help with this.

netCDF is a (the?) standard storage API for geoscientific, weather, climate, satellite data. netCDF supported only one free lossless compressor, and no lossy methods. We incorporated modern lossless codecs and IEEE-754 compatible quantization that can reduce required storage by about 4x and eliminate false precision.

Future Work

- Support S3, the AWS storage scheme (Winter 2024)
- Better support Zarr in **ncclimo**, **ncremap** scripts
- Support Zarr in **ncatted**, **ncrename** binaries
- Propose 4x CMIP6 compression for CMIP7
- Invoke modern codecs in DOE E3SM, NSF CESM (NOAA GFS is done!)
- Support **nccopy** codec API

Acknowledgements

This material is based upon work supported by DOE E3SM (DE-SC0019278), NASA (OSTFL 80NSSC22K1743), and NSF (OAC-2004993). Work performed at UC Irvine, located on the ancestral, unceded homelands of the Acjachemen and Tongva peoples.