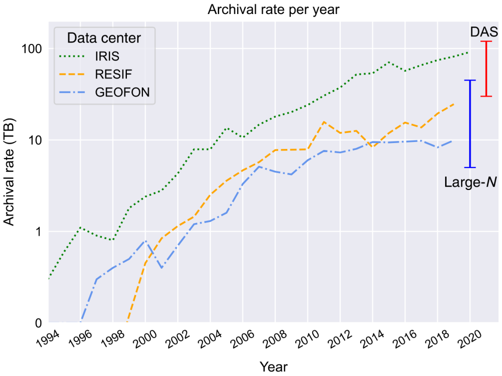
Towards new data archives & distribution mechanisms with object storage

- New types of sensors acquire vast amounts of data
- No standard yet for archival and distribution
- Slowing down science



From Quinteros et al., 2021

Data access from the point of view of scientific users

- large data transfer is challenging to handle through FDSN webservices
- local copies accrue storage costs
- data are not necessarily analysis-ready: often transformed to a different data format before analysis

→ **Joint initiative EarthScope & EIDA**
exploring future storage solutions (adapted to archival and distribution of large data from cloud object storage)

Joint initiative EarthScope & EIDA

- Test different solutions (storage architecture, file formats) for performance and usability (TileDB, zarr, Apache Iceberg)
- Current active involvement EarthScope (Alex Hamilton, Chad Trabant), EIDA nodes GFZ (Javier Quinteros) and EPOS-France (Jonathan Schaeffer, Laura Ermert), Helle Pedersen, Jerry Carter, Angelo Strollo, Philipp Kästli
- Poster (Thursday evening, S02-103): Towards archiving, distributing and using large seismological datasets.

Potential solutions: S3 storage, TileDB, zarr, Apache Iceberg

Common points: Object storage, cloud-friendly, open source, Python APIs, versioning

TileDB

- Multi-D arrays (dense or sparse)
- strong community in life sciences / genomics; main development by commercial company

zarr

- Multi-D arrays (dense)
- community support including Earth science, e.g. accepted format in NASA Earth Science Data Systems, ESA
- not as readily cloud-friendly as TileDB (objects in hierarchical directory). IceChunk may change this in the future

Apache Iceberg

- tabular data, data themselves in Parquet format
- developed by Apache foundation

Examples of reading data from zarr

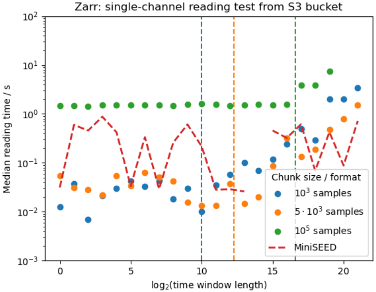
```
import config
import zarr
from obstore.store import S3Store

store = S3Store(
    bucket=config["S3_BUCKET"],
    endpoint=config["S3_SERVICE_URL"],
    aws_access_key_id=config["S3_ACCESS_KEY"],
    aws_secret_access_key=config["S3_SECRET_KEY"],
    virtual_hosted_style_request=False
    )

s3store = zarr.storage.ObjectStore(store)
rows = [0: 86400]
cols = [0: 30]

z = zarr.open_array(s3store, mode="r")
data = z[rows, cols]
```

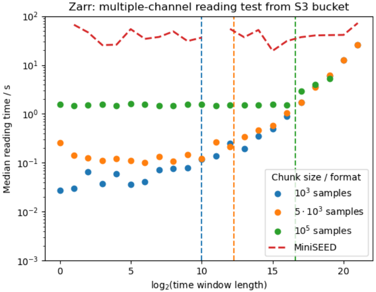
Examples of reading data from TileDB

```python
import config
import tiledb
from isterre_tiledb import isterre_ctx
import numpy as np

ctx = isterre_ctx(user=config["S3_ACCESS_KEY"], pw=config["S3_SECRET_KEY"])
channels = ["XG.01001.00.SPZ","XG.01002.00.SPZ","XG.01003.00.SPZ"]
time_0 = np.datetime64("2020-02-20", "us")
time_1 = np.datetime64("2020-02-21", "us")

array_name = f"{config['S3_DEST_BUCKET']}/nodal.tbd"
array = tiledb.open(array_name, 'r', ctx=ctx)
data = array[channels, time_0: time_1]
```

First test results from GFZ

- Reading samples from a DAS dataset in zarr format stored on a locally hosted S3
- single-channel and multi-channel test
- influence of chunk size tested and compared to reading full miniSEED





Large Dataset Overview

EarthScope
Data Services

GAGE
SAGE

Operated by EarthScope Consortium

ARCO Storage Implementation

Users access archive objects directly

- Expose implementation details
- Decisions about how objects are organized matter
  - Identifiers & semantic meaning in object keys (paths) become important decisions that affect data users

The goal: Analysis Ready Cloud Optimized (ARCO)

It would be best if data providers deliver the data sets in an archive format, adding more complexities to consider.

Operational Requirements

- Authorization granularity
  - Containers for many (millions) of objects
  - Can put data from multiple sources (with different authorization concerns) in the same object
- Transactional isolation granularity
- Time-travel capability
  - Consolidation and vacuuming frequency (metadata only?)
- Efficient access to subsets of the container (“slicing”)
- Language support, project maturity & longevity

EarthScope TileDB Experience

EarthScope uses TileDB for GNSS observables and PPP solutions:

- Two different approaches:
  - GNSS observables live in independent arrays for each station “session”
    - Four dimensions: time, constellation, sat ID, signal code (L1, etc.)
  - PPP solutions live in one array
    - Two dimensions: stream ID, time
- Write frequency matters; streaming data needs to be batched
  - Larger file sizes (~100-500 MB*) result in more efficient reads
  - Compaction/vacuuming essential to maintain performance
    - On the Observables arrays, we only compact metadata
  - Fragmentation across pipelines: golang and python
    - Different compaction/vacuuming implementations & cadences

EarthScope Iceberg Experience

EarthScope prototyping using Iceberg in AWS for new data products

- Collecting real-time streaming data into a table set
  - Testing multiple partitioning schemes
- Hands-off approach (easy):
  - Managed service batches streaming data writes from Kafka to Iceberg
  - Managed service automatic compaction & vacuuming
  - Queries using Athena
- Early experience is very positive, but as an operator it is easy

miniSEED in Iceberg

Goal: implement a temporary buffer of miniSEED data collected as real-time streams in a way that is accessible for fdsnws-dataselect and for an archiving system that makes the final miniSEED repository parts.

Early experiment details:

- Store miniSEED records directly in Iceberg
- Iceberg provides the indexing for efficient access
- Iceberg + managed service provide compaction services
- Data are available with minimum latency

High level points

- Take advantage of broadly accessible, high performance object storage systems
- Avoid points of restriction such as fdsnws-dataselect
- Adopt existing, widely supported data container
- Improved support for more dimensionality
- Support large data set access: for Machine Learning and other large processing cases, highly parallel, read-into-memory in many environments