Optimal Strategies for Storing Earth Science Datasets in the Commercial Cloud

Dieu My T Nguyen[1], Johana Chazaro Cortes[2], Marina M Dunn[3], and Alexey N Shiklomanov[4]

[1]University of Colorado Boulder
[2]California Baptist University
[3]University of California Riverside
[4]NASA Goddard Space Flight Center

March 8, 2023

Impact of Chunk Size on Read Performance of Zarr Data in Cloud-based Object Stores

Dieu My T. Nguyen[1,2], Johana Chazaro Cortes[3,4], Marina M. Dunn[5,6], Alexey N. Shiklomanov[6]

[1]Goddard Earth Sciences Data and Information Services Center, NASA Goddard Space Flight Center, Greenbelt, MD, United States
[2]Adnet Systems, Inc., Bethesda, MD, United States
[3]Science Managed Cloud Environment, NASA Goddard Space Flight Center, Greenbelt, MD, United States
[4]Navteca, LLC, Chevy Chase, MD, United States
[5]Bourns College of Engineering, University of California Riverside, Riverside, CA, United States
[6]Biospheric Sciences Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, United States

Key Points:

- Commercial cloud services offer a more computational and cost-efficient approach to Earth science data management and research.
- Zarr is a powerful, cloud-optimized file format being adopted by the Earth science community for its ability to efficiently handle multi-dimensional datasets.
- Chunking strategy, including chunk shapes and sizes, affected the processing time and memory usage of common data access and analysis operations.
- We found there are trade-offs in performance for time series and spatial operations, but found a middle-range chunking strategy that allows good performance for both operations.

Corresponding author: Dieu My T. Nguyen, `dieumy.t.nguyen@nasa.gov`

Abstract

There is increasing interest in adopting the commercial cloud for Earth science research and data management, given the cost-saving storage, processing, and access efficiencies. In this work, we evaluated the chunking strategies for storing and accessing multi-dimensional datasets in the cloud. We transformed output from the Goddard Earth Observing System model from NetCDF format into a more analysis-ready and cloud-optimized format, Zarr. We explored different strategies for chunking the data into units of different size across the dataset's temporal and spatial dimensions. We compared chunking strategies in terms of their time and memory performance for common data operations, such as extracting a time series at a location or a map at a designated time. The chunking strategy significantly impacted the processing time and memory usage. In general, larger chunks along the target dimension performed best. We found a trade-off in performance between extracting time series and maps—large chunks along the time dimension performed best for extracting time series but worst for maps, and vice versa. However, there were versatile middle-range chunking strategies that performed well for both time-series and spatial access. The chunking strategy also affected rechunking time and dataset compression: smaller chunks required more time to rechunk the original dataset, and larger chunks resulted in smaller datasets. Overall, we provide initial benchmarks and generalized findings focused on optimal chunking strategies for cloud-based storage and usage of large multi-dimensional data.

1 Introduction

While data collection is imperative, the way data is managed and stored can greatly affect how it is used and analyzed (Hey et al., 2009). There is an abundance of high-quality Earth science data across many repositories, but accessing and sharing these datasets can be difficult. The challenge of supporting massive data storage, processing, access, and usage is a long-term bottleneck in Earth science research (Cui et al., 2010; Yang et al., 2011).

Over the last decade, there has been a push to migrate data and models to the commercial cloud (e.g., Amazon Web Services, AWS), which is envisioned to be the computing infrastructure for Earth science research (Yang et al., 2011; Zhuang et al., 2019). A recent example of this migration and assessment of the benefits is a cloud-based data ingest, archive, and management system developed by the Earth Science Data System (ESDS) program at National Aeronautics and Space Administration (NASA) (Ramachandran et al., 2017). Benefits of the commercial cloud include cost savings from flexible and scalable storage and computing options (Ramachandran et al., 2017), and opportunities for cloud-native "analysis-in-place" workflows that preclude the need to download and process large datasets locally (Lynnes & Ramachandran, 2018; Zhuang et al., 2019). While there are discussions across scientific disciplines and in developer communities about adopting the cloud and the challenges associated with cost and shifting the socio-technical paradigm (Lynnes & Ramachandran, 2018), there is a lack of peer-reviewed literature on quantitative and technical assessments of the feasibility of data management, conducting science on the cloud, and general guidance on common requirements.

One feature contributing to the cloud's efficiency and scalability is the object storage architecture (Bucur et al., 2018), such as AWS Simple Storage Service (S3) (*S3*, 2002). In contrast with storage architectures that manage data as file hierarchies or block systems, object storage manages data stores as distinct objects that typically include the data, metadata, and a universally unique identifier (Factor et al., 2005). Unlimited scalability results from the system scaling out by adding storage nodes. Data retrieval in object storage without file structures often may be faster, as each object can be found via its identifying details rather than exact location (Gupta et al., 2020). Thus, object storage on the cloud is suitable for large amounts of Earth science data. However, effective

use of object stores for large multi-dimensional datasets requires careful consideration of data organization and chunking strategy.

Earth science datasets are often distributed as collections of hundreds to thousands of Hierarchical Data Format (HDF; or a derivative format, Network Common Data Form, NetCDF) files (Lucchesi, 2018; Rew & Davis, 1990; The HDF Group, 2000-2010). While this format works well for traditional disk storage of the file system, it is not efficient for cloud object storage (Abernathey et al., 2018). A cloud-friendly alternative is Zarr (Abernathey et al., 2021). Like HDF, Zarr supports an arbitrary number of labeled dimensions and is self-documenting. However, Zarr also offers additional advantages: the metadata (e.g., coordinate system used, date modified, etc.) can be consolidated and kept separate from the data in a single file, and individual chunks are stored as separate objects that can be retrieved independently in a thread-safe manner. Recent work comparing Zarr against a traditional file type (GRIdded Binary Second Edition, GRIB2) used by National Oceanic and Atmospheric Administration (NOAA) found that Zarr was significantly more efficient (e.g., 40x faster time-series access) on both smaller and high-performance computing nodes (Gowan et al., 2022).

Most modern multi-dimensional array storage formats—including HDF5, NetCDF, TIFF, and Zarr—allow control of how the data is stored and compressed, choices which can have significant performance implications (Kang et al., 2020). For example, when the entire dataset is serialized into a single block, a subsetting operation must read the entire dataset into memory, which is wasteful. In contrast, when data is stored in smaller chunks that can be read and written individually, subsetting operations only need to read the chunks relevant to the requested subset, dramatically reducing I/O times (Rew, 2013; Kang et al., 2020). However, this must be balanced against the overhead of each individual read operation; for the same total amount of data read, fewer reads from larger files will be faster than more reads from smaller files (Kang et al., 2020). Maximizing application performance requires careful attention to chunking strategy, including total chunk size and how this is distributed across dimensions.

In this paper, we explored the performance of various strategies for chunking a multi-dimensional Earth science dataset by comparing the processing time and memory usage for common data access operations using a single processor to resemble simplified use cases. We expected the best strategies will differ for each type of access, as different dimensions need to be accessed for the appropriate operations. However, we also expected to find a range of chunking strategies that provide reasonable performance for a variety of analyses. We provide comprehensive and quantitative benchmarks and general guidance for the optimal chunking strategies for cloud storage of Earth science data.

2 Methods

2.1 Dataset

This study focused on outputs from the Goddard Earth Observing System (GEOS) model (Rienecker et al., 2008), which simulates various meteorological, climatological, and atmospheric chemistry processes. Among many other applications, GEOS is the model underlying the Modern-Era Retrospective Analysis for Research and Applications (MERRA) reanalysis product (Rienecker et al., 2011). GEOS outputs include over 100 variables distributed over multiple output streams, depending on the configuration. For this work, we focused on one variable—Black Carbon Extinction Aerosol Optical Thickness (BCEXTTAU)—from the Aerosol Diagnostics outputs (tavg1_2d_aer_Nx; M2T1NXAER) from the GEOS Forward Processing (GEOS-FP) Data Assimilation System (DAS). The complete GEOS-FP DAS output is available in its original format from the NASA Center for Climate Simulations (NCCS) data portal (https://portal.nccs.nasa.gov/datashare/gmao/geos-fp/das/).

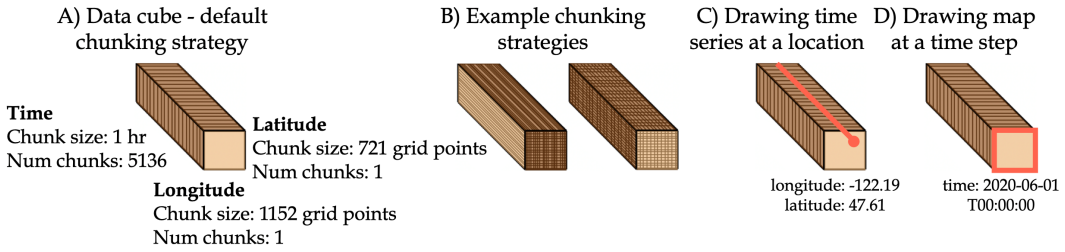

Figure 1. Conceptual multi-dimensional dataset and chunking. A) The dataset represented as a data cube with default dimensions for time, longitude, and latitude. B) Various chunking strategies affect chunk sizes in each dimension. C) Extracting a time series at a location (e.g., longitude: -122.19, latitude: 47.61) as a line through the time dimension. D) Extracting a map at a time step (e.g., 2020-06-01 T00:00:00) as a plane through a single time slice.

The original GEOS-FP output was produced and distributed in NetCDF-4 format, with 1 global file per timestep. We transformed the dataset into Zarr archives. The data was compressed using Zarr's default compressor, Blosc, a meta-compressor that uses the Zstandard (Zstd) algorithm, and results in level-3 compression. GEOS-FP contains 3-dimensional fields (longitude, latitude, time), with a 1-hour reporting interval and spatial resolution of 0.3125x0.25 degrees. The spatial bounds are -180 to 179.6875 degrees longitude and -90 to 90 degrees latitude. Our combined dataset had a dimensionality of 5136 (time) x 1152 (longitude) x 721 (latitude) (Lucchesi, 2018). The default chunking strategy of this dataset was: 5136 chunks (chunk size: 1) in time, 1 chunk (chunk size: 1152) in longitude, and 1 chunk (chunk size: 721) in latitude (Figure 1A). Data was stored in an AWS S3 bucket.

2.2 Chunking strategies

We performed this study on a high-performance computing cluster set up on AWS. Each partition contained 30 compute nodes (c5n.18xlarge) with 36 Intel Xeon Platinum 8124M 3.00GHz CPU cores, with 192GB of RAM. All code was developed in Python (version 3.9.7). We used the Python package Xarray (0.19.0) (Hoyer & Hamman, 2017) for working with labeled multi-dimensional data arrays and matplotlib (3.4.3) (Hunter, 2007) for plotting.

We tested various chunking strategies, comparing performance for extracting time series and maps. We used the Python package Rechunker (0.4.2), which allows efficient and scalable manipulation of the chunk structure of Zarr datasets while preserving the integrity of the underlying data (Augspurger & Abernathey, 2020). We rechunked the data with various chunk sizes across longitude, latitude, and time (Figure 1B). Specifically, we tested the following dimensional chunk sizes:

- Time: 1, 6, 12, 24, 48, 120, 720, 1440, 2160, 5136
- Longitude: 10, 50, 100, 1152
- Latitude: 10, 50, 100, 721

2.3 Experiments and performance metrics

We measured how each chunking strategy performed on two common operations: 1) extracting a time series at a coordinate, and spatially aggregated over a geographic region; and 2) drawing a map at a specific datetime, and averaged over a temporal range (Figure 1C). For each operation, our benchmark included both metadata querying and

the actual reading of the data into memory. We used Xarray's `sel()` and `isel()` methods to "lazily" select the data (i.e. reading metadata and building a graph), and then used the Dask `compute()` method to trigger the actual loading of the array as follows (Hoyer & Hamman, 2017):

```
# Time series at a coordinate
data.sel(lat=47.61, lon=-122.19).compute()
# Map at a datetime
data.sel(time='2020-06-01').isel(time=0).compute()
# Time series over a spatially-averaged region (Ohio, USA)
data.sel(lat=slice(38.21,42.25), lon=slice(-84.91,-80.5)).mean(['lat','lon']).compute()
# Map over a temporally-averaged time range (6-hour range)
data.sel(time=slice('2020-06-01T00','2020-06-01T05')).mean(['date']).compute()
```

We performed these access operations using the output datasets produced by the chunking strategies described above, and tracked their performance via several metrics. We aimed to purely test the I/O component of data access, without the added complication and overhead associated with parallel execution. Therefore, we only used a single core for reading any particular Zarr archive.

To avoid AWS caching issues where subsequent accesses are faster than the first (see SI), we measured performance for only the first access. Time metrics allowed us to assess the usability of the chunking strategies. We measured wall clock time using the built-in Python module "time" (Van Rossum & Drake, 2009) as follows:

```
start = time.time()
data.sel(lat=47.61, lon=-122.19).compute()
wall_time = time.time() - start
```

To compare wall times of various strategies, we defined the performance bias metric as the largest wall time divided by smallest wall time, rounded to the nearest whole number. In addition, we reported the processing speed (data points per second) for each operation, calculated as length of data array / wall time. We also measured central processing unit (CPU), and present these measurements in Supplementary Information Figure S4.

Peak memory usage informs us of the resource requirements needed for the operations. We used the memory_usage() method in the Python module "memory-profiler" to output the memory (in MiB) used throughout the execution time over default intervals of 0.1 sec (Pedregosa & Gervais, 2021), extracting the maximum memory as the peak memory. We also computed the peak memory usage per data point (peak memory / length of data array) as an indicator of "memory efficiency."

We reported two other relevant metrics: (1) the time taken to rechunk the dataset for each strategy, found by measuring the wall time for the entire rechunking pipeline (including reading the default Zarr data archive, executing the rechunker, consolidating the metadata, and saving the new Zarr archive to S3); and (2) the storage size of each rechunked dataset stored on S3.

3 Results

3.1 Trade-offs between time-series and spatial access

Regarding the wall time for extracting a time series, larger time chunks and smaller spatial chunks yielded the best performance (lowest wall time and highest processing speed) (Figures 2A, Supplementary Information S1A). Wall time ranged from 0.065 sec (chunk

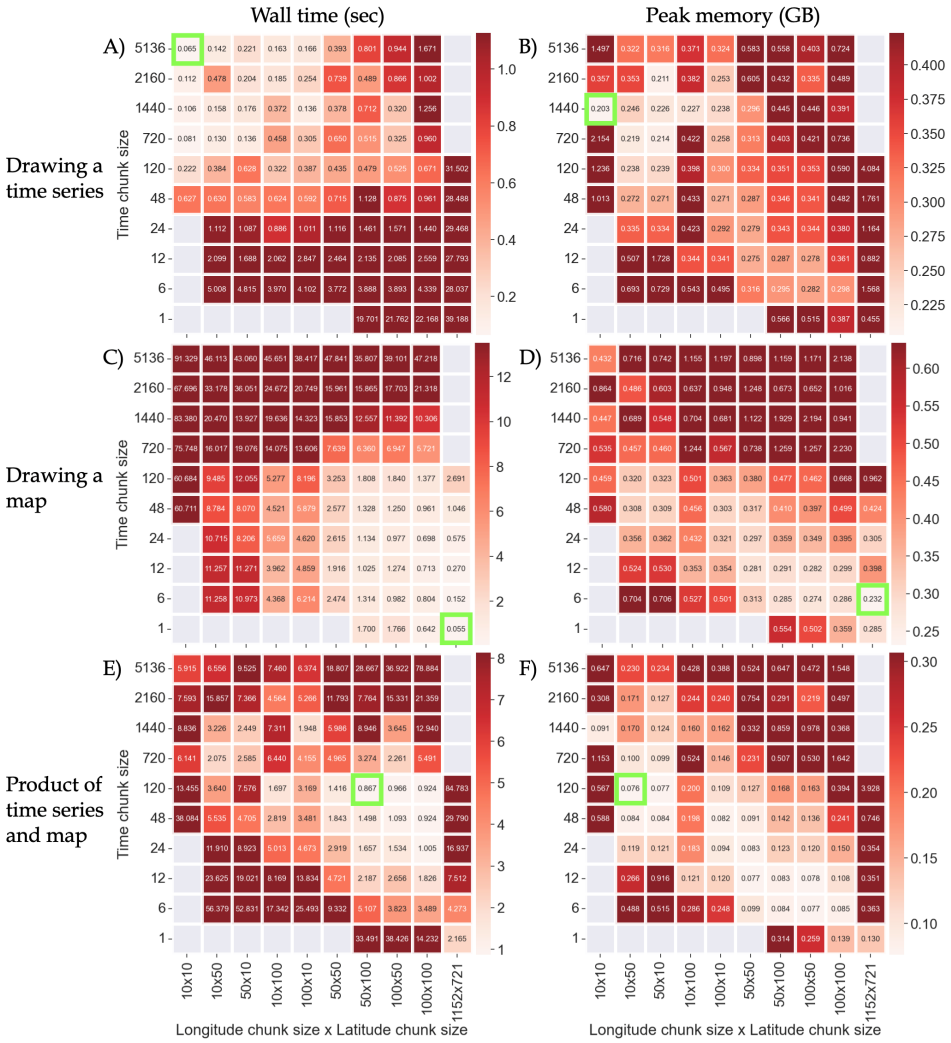

Figure 2. Wall time and peak memory consumption by chunking strategy. Green boxes highlight optimal strategies. Gray squares represent strategies omitted as they yield high rechunking time and are not as realistic for common dataset usage. A-B) Heatmaps of wall time (seconds) and peak memory (gigabytes) consumption of chunking strategies when drawing a time series at a single coordinate. C-D) Heatmaps when drawing a map for a single timestep. E-F) Heatmaps of the product of time series and map operations to show general efficient chunk size guidance when drawing both time series and maps.

size: 5136 time x 10 longitude x 10 latitude) to 39.188 sec (1x1152x721). Meanwhile, for extracting a map, smaller time chunks and larger spatial chunks yielded the best performance (Figures 2C, S1C). Wall time ranged from 0.055 sec (1x1152x721) to 91.329 sec (5136x10x10). The optimal strategy for accommodating both access pattern types had intermediate time and spatial chunk sizes, and had a product wall time of 0.867 sec (120x50x100, Figure 2E). The highest average speed is 7.5×10^6 data points per sec (1x1152x721, Figure S1E). The optimal strategy for time-series access contiguous in the time dimension (5136x10x10) was 91.329 sec or 1405 times slower for extracting a map (Table 1). The optimal strategy for a map containing contiguous chunks in the spatial dimensions (1x1152x721) was 39.188 sec or 713 times slower for extracting a time series. The strat-

Chunking strategy (time x longitude x latitude chunk)	**Time series wall time (sec)**	**Map wall time (sec)**	**Performance bias (highest/lowest)**
Optimal for time access Contiguous in time dimension (5136x10x10)	0.065	91.329	1405
Optimal for spatial access Contiguous in spatial dimensions (1x1152x721)	39.188	0.055	713
Optimal for time and spatial access (120x50x100)	0.479	1.808	4

Table 1. Comparing performance of optimal chunking strategies

egy optimal for both access patterns (120x50x100) took 0.479 sec for a time series and 1.808 sec for a map, with a low performance bias of 4. Finally, when correlating chunk size in memory with processing speed for each access pattern, we found positive correlations for both (Figure 3A). The highest speed was 1.50×10^7 sec for chunk size 3.32×10^6 B or 3.32 MB.

Regarding peak memory consumption when extracting a time series, strategies with larger time chunks and smaller spatial chunks produced the lowest peak memory overall and per data point (Figures 2B, S1B). Peak memory ranged from 0.203 GB (1440x10x10) to 4.084 GB (120x1152x721). For extracting a map, strategies with larger spatial chunks and smaller time chunks produced the lowest peak memory (Figure 2D, S1D). Peak memory ranged from 0.232 GB (6x1152x721) to 2.230 GB (720x100x100). Similar to wall time, optimal strategies for the product of the two operations consisted of intermediate chunks in the three dimensions (Figure 2F). The lowest product peak memory was 0.076 GB (120x10x50), and the lowest average peak memory per data point was 0.019 MiB (1440x10x10, Figure S1F). For the time-optimized strategy for time-series access (5136x10x10), the peak memory was 1.497 GB, or 1.294 GB higher than the memory-optimized strategy (1440x10x10). For the time-optimized strategy for spatial access (1x1152x721), the peak memory was 0.285 GB, or 0.053 GB higher than the memory-optimized strategy (6x1152x721). While there was no correlation between chunk size in memory and peak memory usage per data point for individual operations, combining them produced a negative correlation (Figure 3B). The overall lowest memory usage per data point was 2.66×10^{-4} MiB for chunk size 3.32 MB—the same chunk size for the highest speed above-mentioned. Additionally, we observed a negative correlation between peak memory per data point and processing speed (Figure 3C).

3.2 Chunking strategy affects rechunking time and storage size

To further assess the usability of chunking strategies and the tractability of changing default storage layouts, we tracked the rechunking time. Rechunking the dataset into smaller chunks across all dimensions consumed the most time (bottom left corner, Figure 4A). On the other hand, the more data points grouped into a chunk (i.e., larger chunk sizes), the quicker the rechunking process. We observed a wide range of rechunking times, varying from 0.111 hours or 6.66 minutes (5136x100x100) to 46.516 hours (6x50x10). The quickest rechunking strategy (5136x100x100) had high wall time and peak memory for both tasks (48.408 sec and 1.626 GB, respectively). The time-optimized strategy supporting multiple access patterns with low performance bias (48x100x100) took 0.374 hours or 22.44 minutes to rechunk.

Chunking strategies also affected the output archive storage size. Most chunking strategies resulted in fairly similar archive storage sizes (Figure 4B). However, strategies with large or contiguous spatial dimensions resulted in archive sizes at least 1 GB or 10% smaller than the strategy with the largest archive (size: 9.824 GB, 40x10x10) (Fig-

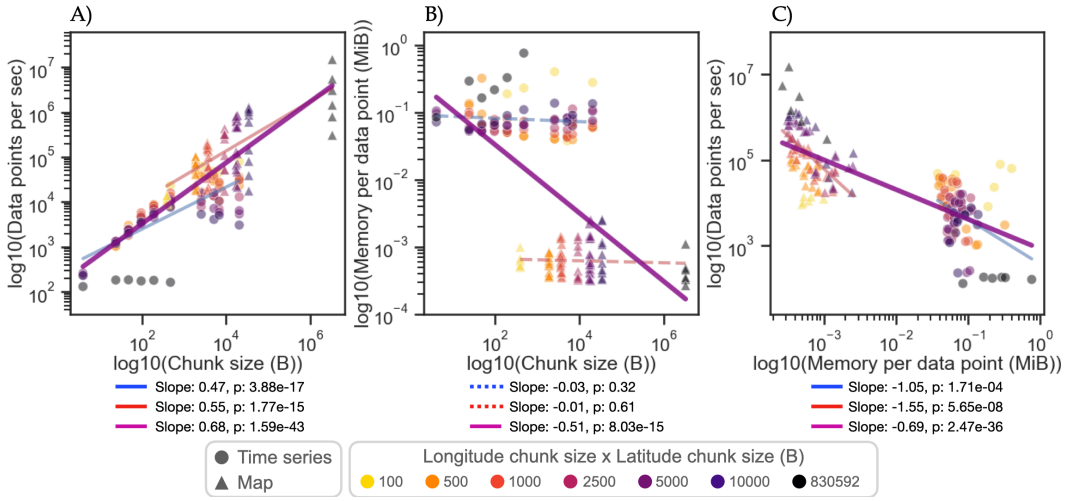

Figure 3. The relationship between performance metrics. Data presented in the heatmaps in Figure S1 was gathered into seven groups based on the products of longitude and latitude chunk sizes. Circles represent time series operations and triangles represent map operations. Solid lines represent $p \leq 0.05$; dashed lines represent $p > 0.05$). A) Scatter plot of the log of the number of data points processed per second (processing speed) as a function of the log of chunk size (bytes) and the best fit line on all data. B) Scatter plot and best fit line of the log of peak memory usage per data point (memory demand in mebibytes) as a function of the log of chunk size. C) Scatter plot and best fit line of the log of the processing speed as a function of the log of the peak memory demand per data point.

ure 4B, 1152x721 column). The time-optimized chunking strategy (48x100x100) resulted in an intermediate archive size of 9.137 GB.

4 Discussion

In line with similar studies for HDF formats (Rew, 2013; Jelenak, 2014), our results showed that optimal chunking strategies vary with access pattern, evidenced by trade-offs in performance between time series and spatial access. When accessing data along a particular dimension, larger chunks or a contiguous block in that dimension allow higher performance gains, consistent with analytical models (Kang et al., 2020). Our results indicated that the best strategy for extracting a time series contained larger time chunks and smaller spatial chunks, while the best strategy for extracting a map contained smaller time chunks and larger spatial chunks. With smaller chunks, more bits of data need to be accessed in each I/O operation, and more time and instructions are needed to look up all chunks (Worringen et al., 2003). Larger chunks or a contiguous block reduce the number of chunks to find and load, decreasing processing time.

However, there might be diminishing returns for large chunks when considering memory usage. While wall time is important for responsiveness and usability, memory constraints determine whether an operation can be carried out at all without crashing (and often killing the parent process). Chunks are all-or-nothing—the entire chunk must be read into memory and uncompressed to operate on any of its data points. In our experiments, the optimal strategy based on wall time for time series or spatial access had large chunks along the respective dimension(s), but was not the least memory-intensive. Choice of chunking strategy therefore needs to consider memory limitations of user systems; chunks

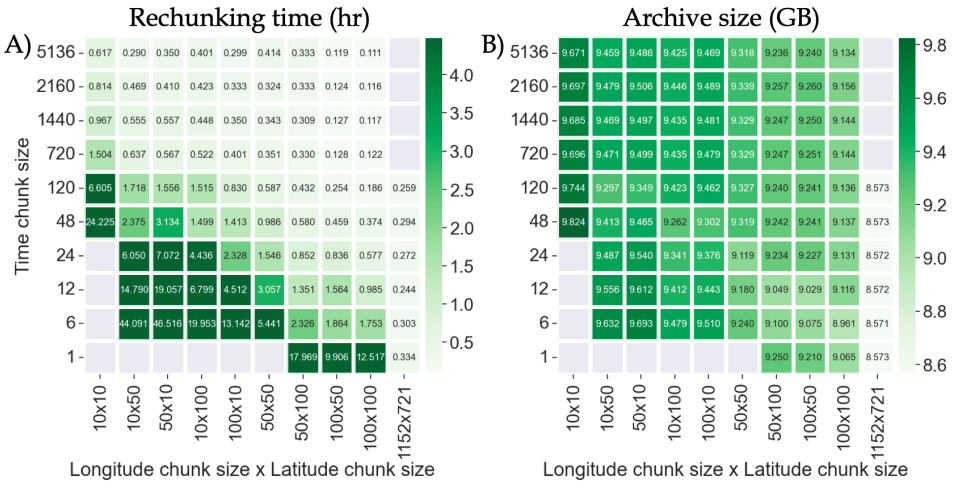

Figure 4. The rechunking time and output archive size of different chunking strategies. Gray squares represent omitted strategies. A) The time (hours) required to process the rechunking strategies. B) Impact of respective chunking strategy on file size (gigabytes).

should be large enough to minimize processing time, but small enough to fit into memory.

While organizing multi-dimensional data into a contiguous block yields optimal performance for dimension-targeted access, many applications require datasets to support multiple different access patterns (especially for larger datasets, where creating copies of multiple datasets with different strategies is expensive or impractical). To that end, we identified the optimal strategy by wall time for extracting both time series and maps: splitting data in all dimensions into intermediate-sized bits (48x100x100). This strategy was much slower than the optimal strategy for each access pattern (e.g., 1.192 sec vs. 0.020 sec for time-series access). However, performance improvements in user interfaces have diminishing returns, and this delay may be acceptable. For example, the 1.192 sec response here is well within the 2 second limit found in a study of user tolerance for responsivity in a web application (Nah, 2004). This is especially the case given the dramatic improvement in support performance across multiple access patterns.

We also considered optimal chunk sizes in terms of their memory footprint. We initially consulted with groups in the geospatial developer community, such as Pangeo and CarbonPlan, and compared our results to their recommendations. These recommendations range from 1 MB uncompressed (Miles et al., 2021), to 10 MB - 1 GB (Rocklin, 2015), to 50 MB - 100 MB (Signell, 2020) for common data operations. Though this guidance provides suggestions for the overall chunk size, it does not provide recommendations for how chunks should be distributed across dimensions. Our findings show that the optimal chunk sizes ($\sim$3.32 MB) for this case align most closely with the $\sim$1 MB (uncompressed) recommendation, but also most importantly provides more detailed information on the performance implications of different chunking strategies across various dimensions.

Previously, other projects such as Pangeo and Open Data Cube have shown that the most effective way to read data is to use chunks 10 to 100 MB in size and to use multiple machines to access object storage simultaneously (e.g., via a Kubernetes cluster), thereby avoiding the limitation of I/O bandwidth of a single machine. In our study, our aim was to understand the effect of chunking in simple use cases of geospatial data ac-

cess, common for Earth Science scientists who may be new to the commercial cloud. Thus, we simplify our tests to only the I/O components of data access without additional setup and overhead from parallel execution. Although common for more advanced users — and greatly facilitated through the native integration of parallel computing libraries like Dask into data processing libraries like Xarray — parallel computing is often challenging for less technical users, and can be impossible in strictly managed, cost-controlled cloud computing deployments for data exploration (e.g., mybinder.org; NASA's Multi-Mission Algorithm Analysis Platform, MAAP) or "serverless" compute capabilities like Amazon Web Services Lambda. Deploying a Kubernetes cluster in particular requires a great deal of specific technical expertise that very few scientists typically have. That said, we recognize the interest and benefits of using parallel computing for cloud-native data access and analysis, and we recommend a broader, more holistic study that factors in parallelization for future work.

We also assessed how chunking strategy affects rechunking time and archive size, which may be important in some data management contexts. From the default dataset contiguous in the spatial dimensions, rechunking took the longest when all chunk sizes were small, as many chunks need to be written and compressed. Rechunking time widely ranges from as little as ∼6.66 min for strategies with larger chunks in all dimensions to as long as ∼46 hours with smaller chunks. For near-real-time and operational data pipelines, such as NASA's Land Atmosphere Near real-time Capability for Earth Observing System (LANCE) data products that are available within 3 hours from satellite observations (NASA EOSDIS, 2019), and especially NOAA's GOES satellites that produces critical weather data every 5 - 15 minutes (NOAA, n.d.), time invested in rechunking data to facilitate analysis needs to be carefully weighted against latency requirements. Our results show strategies with the smallest rechunking times may not be optimized for multiple access patterns, but the optimal strategy's rechunking time of 22.44 minutes may be tolerable for some applications. Also, time invested in rechunking may be offset by accelerated product generation workflows, where such workflows are limited by I/O performance. Archive size is also important for large Earth science datasets and limited storage quotas. Consistent with literature (Tang et al., 2021), our results showed that larger chunks have a greater compression effect, as they contain more repeated data patterns compression libraries can target (Lelewer & Hirschberg, 1987; Fitriya et al., 2017). Although compression efficiency decreased with smaller chunks, we gained higher speed and access versatility.

Although the exact storage strategy specifics will vary by dataset, trends observed in this work can be generalized for other applications. Further studies are needed on efficient chunking for other types of data encountered by Earth scientists, including sparse gridded datasets, such as active fire detections from Moderate Resolution Imaging Spectroradiometer (MODIS) (Justice et al., 2002). Trade-offs in chunking strategies might become more significant for higher dimensional datasets, such as vertically-resolved gridded time series produced by the GEOS Composition Forecasting (CF) model (Keller et al., 2021). Another future direction is tracking performance for spatio-temporal analyses simultaneously spanning multiple dimensions; namely, where data across every dimension may be needed (such as in Empirical Orthogonal Function, EOF, analysis), performance may depend primarily on chunk size but not chunk distribution. We present initial results as benchmarks for future studies on multi-dimensional Earth science or general datasets that use cloud-optimized formats as part of a larger community effort to establish scalable computing infrastructure supporting better scientific discovery and data utilization.

5 Availability Statement

The GEOS-FP data used in the study was sourced from NASA's GMAO Near-Real Time Data Products available at: `https://gmao.gsfc.nasa.gov/GMAO_products/NRT`

_products.php. All Python code is available at the GitHub repository under MIT License: https://github.com/dieumynguyen/ZarrOptimalStorage. DOI: 10.5281/zenodo.7422933 (Nguyen, 2022).

6 Author Contributions

ANS conceived and supervised this project. DMTN and MMD wrote the initial computational framework and performed initial experiments and analyses. DMTN performed the remainder of the project's computational framework design, validation, analysis, and visualization. DMTN, JCC, MMD, and ANS wrote, reviewed, and edited the manuscript.

Acknowledgments
This work was supported by Earth Information System (EIS) funding from NASA headquarters. We thank Aimee Barciauskas, the EIS Fire team, the NASA Goddard Earth Sciences Data and Information Services Center (ESDIS), and the Goddard Earth Sciences Data and Information Services Center (GESDISC) teams for insightful discussions. We also thank Christine E. Smit and Hailiang Zhang for reading the manuscript and providing feedback. The authors declare that they have no conflict of interest.

References

Abernathey, R. P., Augspurger, T., Banihirwe, A., Blackmon-Luca, C. C., Crone, T. J., Gentemann, C. L., ... others (2021). Cloud-native repositories for big scientific data. *Computing in Science & Engineering*, *23*(2), 26–35.

Abernathey, R. P., Hamman, J., & Miles, A. (2018). Beyond netcdf: Cloud native climate data with zarr and xarray. In *Agu fall meeting abstracts* (Vol. 2018, pp. IN33A–06).

Augspurger, T., & Abernathey, R. (2020). *Rechunker — rechunker 0.4.3.dev5+g0a0d1eb documentation.* Retrieved from https://rechunker.readthedocs.io/

Bucur, V., Dehelean, C., & Miclea, L. (2018). Object storage in the cloud and multi-cloud: State of the art and the research challenges. In *2018 ieee international conference on automation, quality and testing, robotics (aqtr)* (pp. 1–6).

Cui, D., Wu, Y., & Zhang, Q. (2010). Massive spatial data processing model based on cloud computing model. In *2010 third international joint conference on computational science and optimization* (Vol. 2, pp. 347–350).

Factor, M., Meth, K., Naor, D., Rodeh, O., & Satran, J. (2005). Object storage: The future building block for storage systems. In *2005 ieee international symposium on mass storage systems and technology* (pp. 119–123).

Fitriya, L., Purboyo, T., & Prasasti, A. (2017, Jan). A review of data compression techniques. *International Journal of Applied Engineering Research*, *12*, 8956-8963.

Gowan, T., Horel, J., Jacques, A., & Kovac, A. (2022, Jan). Using cloud computing to analyze model output archived in zarr format. *Journal of Atmospheric and Oceanic Technology*. Retrieved from https://journals.ametsoc.org/view/journals/atot/aop/JTECH-D-21-0106.1/JTECH-D-21-0106.1.xml doi: 10.1175/JTECH-D-21-0106.1

Gupta, A., Mehta, A., Daver, L., & Banga, P. (2020). Implementation of storage in virtual private cloud using simple storage service on aws. In *2020 2nd international conference on innovative mechanisms for industry applications (icimia)* (pp. 213–217).

Hey, A. J., Tansley, S., Tolle, K. M., et al. (2009). *The fourth paradigm: data-intensive scientific discovery* (Vol. 1). Microsoft research Redmond, WA.

Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled arrays and datasets

in Python. *Journal of Open Research Software*, *5*(1). Retrieved from http://doi.org/10.5334/jors.148 doi: 10.5334/jors.148

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. doi: 10.1109/MCSE.2007.55

Jelenak, A. (2014, Dec). Organizing data to support diverse access patterns. Agu. Retrieved from https://agu.confex.com/agu/fm14/webprogram/Paper26418.html

Justice, C., Giglio, L., Korontzi, S., Owens, J., Morisette, J., Roy, D., ... Kaufman, Y. (2002). The modis fire products. *Remote Sensing of Environment*, *83*(1), 244-262. Retrieved from https://www.sciencedirect.com/science/article/pii/S0034425702000767 doi: 10.1016/S0034-4257(02)00076-7

Kang, D., Rübel, O., Byna, S., & Blanas, S. (2020). Predicting and comparing the performance of array management libraries. In *2020 ieee international parallel and distributed processing symposium (ipdps)* (pp. 906–915).

Keller, C. A., Knowland, K. E., Duncan, B. N., Liu, J., Anderson, D. C., Das, S., ... Pawson, S. (2021, Apr). Description of the NASA GEOS composition forecast modeling system GEOS-CF v1.0. *Journal of Advances in Modeling Earth Systems*, *13*(4). Retrieved from https://doi.org/10.1029/2020ms002413 doi: 10.1029/2020ms002413

Lelewer, D. A., & Hirschberg, D. S. (1987, Sep). Data compression. *ACM Computing Surveys*, *19*(3), 261–296. Retrieved from https://doi.org/10.1145/45072.45074 doi: 10.1145/45072.45074

Lucchesi, R. (2018, July). *File specification for geos fp (forward processing)* (Tech. Rep.). NASA Goddard Space Flight Center Global Modeling and Assimilation Office. Retrieved from https://gmao.gsfc.nasa.gov/pubs/docs/Lucchesi1203.pdf

Lynnes, C., & Ramachandran, R. (2018). Generalizing a data analysis pipeline in the cloud to handle diverse use cases in nasa's eosdis. In *Igarss 2018-2018 ieee international geoscience and remote sensing symposium* (pp. 422–425).

Miles, A., Jakirkham, Bussonnier, M., Moore, J., Fulton, A., Bourbeau, J., ... Hunt-Isaak, I. (2021). *zarr-developers/zarr-python:.* Zenodo. Retrieved from https://zenodo.org/record/5712786 doi: 10.5281/ZENODO.5712786

Nah, F. F.-H. (2004). A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153–163.

NASA EOSDIS. (2019). *Land, Atmosphere Near real-time Capability for EOS (LANCE).* NASA's Earth Observing System Data and Information System (EOSDIS). Retrieved from https://earthdata.nasa.gov/earth-observation-data/near-real-time/

Nguyen, D. M. T. (2022). *Data and code repository for the study on optimal strategies for storing earth science datasets in the commercial cloud.* Retrieved from https://github.com/dieumynguyen/ZarrOptimalStorage

NOAA. (n.d.). *Goes overview.* Retrieved from https://www.noaasis.noaa.gov/GOES/goes_overview.html

Pedregosa, F., & Gervais, P. (2021, Dec). *memory-profiler: A module for monitoring memory usage of a python program.* Retrieved from https://pypi.org/project/memory-profiler/

Ramachandran, R., Baynes, K., Murphy, K., Jazayeri, A., Schuler, I., & Pilone, D. (2017). Cumulus: Nasa's cloud based distributed active archive center prototype. In *2017 ieee international geoscience and remote sensing symposium (igarss)* (pp. 369–372).

Rew, R. (2013, Jan). *Chunking data: Why it matters.* Retrieved from https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking_data_why_it_matters

Rew, R., & Davis, G. (1990). Netcdf: an interface for scientific data access. *IEEE computer graphics and applications*, *10*(4), 76–82.

Rienecker, M. M., Suarez, M., Todling, R., Bacmeister, J., Takacs, L., Liu, H., ... others (2008). *The geos-5 data assimilation system: Documentation of versions 5.0.1, 5.1.0, and 5.2.0* (Tech. Rep.). Retrieved from `https://gmao.gsfc.nasa.gov/pubs/docs/GEOS-5.0.1_Documentation_r3.pdf`

Rienecker, M. M., Suarez, M. J., Gelaro, R., Todling, R., Bacmeister, J., Liu, E., ... others (2011). Merra: Nasa's modern-era retrospective analysis for research and applications. *Journal of climate*, *24*(14), 3624–3648.

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference.*

S3. (2002). Amazon Web Services, Inc. Retrieved from `https://aws.amazon.com/s3/`

Signell, R. (2020, Feb). *Cloud-performant reading of netcdf4/hdf5 data using the zarr library.* Retrieved from `https://medium.com/pangeo/cloud-performant-reading-of-netcdf4-hdf5-data-using-the-zarr-library-1a95c5c92314`

Tang, H., Byna, S., Petersson, N. A., & McCallen, D. (2021). Tuning parallel data compression and i/o for large-scale earthquake simulation. In *2021 ieee international conference on big data (big data)* (p. 2992-2997). doi: 10.1109/BigData52589.2021.9671876

The HDF Group. (2000-2010). *Hierarchical data format version 5.* Retrieved from `http://www.hdfgroup.org/HDF5`

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual.* Scotts Valley, CA: CreateSpace.

Worringen, J., Traff, J., & Ritzdorf, H. (2003). Fast parallel non-contiguous file access. In *Sc '03: Proceedings of the 2003 acm/ieee conference on supercomputing* (p. 60-60). doi: 10.1145/1048935.1050211

Yang, C., Goodchild, M., Huang, Q., Nebert, D., Raskin, R., Xu, Y., ... Fay, D. (2011, July). Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, *4*(4), 305–329. Retrieved from `https://doi.org/10.1080/17538947.2011.587547` doi: 10.1080/17538947.2011.587547

Zhuang, J., Jacob, D. J., Gaya, J. F., Yantosca, R. M., Lundgren, E. W., Sulprizio, M. P., & Eastham, S. D. (2019). Enabling immediate access to earth science models through cloud computing: application to the geos-chem model. *Bulletin of the American Meteorological Society*, *100*(10), 1943–1960.