OGC® DOCUMENT: 24-044

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t20-D030

Open Geospatial Consortium

TESTBED 20 HIGH-PERFORMANCE GEOSPATIAL COMPUTING OPTIMIZED FORMATS REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2025-02-13
Approval Date: 2025-03-06
Publication Date: 2025-04-04
Editor: Eugene Yu, Liping Di

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

I KEYWORDS

The following are keywords to be used by search engines and document catalogues.

hackathon, application-to-the-cloud, testbed, docker, web service

II CONTRIBUTORS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Eugene Yu	George Mason University	Editor
Liping Di	George Mason University	Editor
Sina Taghavikish	Open Geospatial Consortium	Task Lead
Michael Leedahl	Maxar Technologies Inc	Contributor
Furqan Baig	University of Illinois Urbana-Champaign	Contributor
Gérald Fenoy	Geolabs	Contributor
Carl Reed	Carl Reed and Associates	Content Reviewer

III OVERVIEW

High-Performance Geospatial Computing (HPGC) is crucial for addressing the increasing complexity and volume of geospatial data across sectors such as meteorology, disaster management, and climate change. While High Performance Computing (HPC) offers significant potential, integrating it with geospatial tools remains a challenge. The OGC Testbed 20 High-Performance Geospatial Computing Optimized Formats Report (this Report) investigates the potential of cloud-optimized formats (such as Cloud Optimized GeoTIFF (COG), GeoZarr, and GeoParquet) to optimize HPGC performance by addressing data transfer bottlenecks and enabling efficient data access. By evaluating these formats and exploring data indexing

and partitioning strategies, the goal is to develop a framework for scalable and efficient HPGC workflows. The Testbed found that using Cloud-Optimized GeoTIFFs (COGs) in HPC environments offers efficient data access and flexible data management, but balancing file sizes and read requests can affect performance. GeoZarr provides scalable data management and efficient parallel read/write operations, though it generates many files that can cause I/O issues in traditional block-storage systems. GeoParquet's columnar layout and multithreading capabilities enhance storage efficiency and data retrieval, but the lack of a spatial index and the need to decompress entire chunks can complicate data management.

IV FUTURE OUTLOOK

The Report anticipates future advancements in HPGC to focus on geospatial data indexing and partitioning, both of which are critical for unlocking the full potential of parallelism in HPGC. This Report also proposes key research questions for future investigation, including the potential use of existing cloud-optimized formats as HPC-optimized formats, best practices for geospatial data indexing and partitioning, and optimal solutions for an HPC-optimized format. The evaluation of cloud-optimized formats and their applicability to HPGC in Testbed-20 may lead to the development of new HPC-optimized formats. The future of High-Performance Geospatial Computing (HPGC) looks promising, with advancements in HPC and cloud-optimized formats enhancing geospatial data processing and analysis. Key research areas include leveraging new features in Zarr version 3, improving GeoParquet libraries, and exploring adaptive indexing and partitioning strategies to optimize performance in HPC environments.

V VALUE PROPOSITION

The research conducted in the Testbed 20 HPGC activity underscored the value of integrating cloud-optimized formats within HPC environments in order to support geospatial workflows more effectively, improve computational efficiency, and advance the use of parallelism in HPGC implementations. The OGC Testbed-20 activity built on previous work in OGC Testbed-17[3] and OGC Testbed-19[29], which advanced data encoding standards and developed an API-based approach for HPGC, respectively. Through the optimization of data formats and management practices, acceleration in time-to-insight can be achieved, enabling faster and more informed decision-making. The findings documented in this Report contribute to the development of a robust HPGC ecosystem, empowering researchers and practitioners to tackle complex geospatial challenges with increased efficiency and effectiveness. The Testbed found that Cloud-Optimized GeoTIFFs (COGs) offer efficient data access and flexible management through internal compression and optimized overviews. GeoZarr supports scalable data processing and efficient parallel read/write operations, leveraging HPC capabilities and offering versatile storage options. GeoParquet enhances storage efficiency and data retrieval with its columnar layout and multithreading capabilities, while also being compatible with existing tools and workflows.

1 INTRODUCTION

1 INTRODUCTION

The increasing volume and complexity of geospatial data necessitates the application of High-Performance Computing (HPC) for efficient analysis and processing. The use of HPC in fields such as meteorology, disaster management, and climate change are enabling unprecedented insights and rapid decision-making. While HPC offers immense potential for accelerating geospatial computations, its integration with traditional geospatial tools and workflows often proves challenging. This OGC Testbed-20 Report delves into the pivotal role of HPC in the geospatial sector. The Report explores the challenges and advancements in optimizing High-Performance Geospatial Computing (HPGC) resources for analytical applications, with a focus on the integration of cloud-optimized formats within HPC environments.

1.1. Aims

The primary aim of the Testbed-20 HPGC activity was to investigate the potential of integrating cloud-optimized formats within HPC environments. This integration is expected to support geospatial workflows more effectively, improve computational efficiency, and advance the use of parallelism in HPGC implementations. The HPGC activity also aimed to explore the potential of existing cloud-optimized formats as HPC-optimized formats and the role of robust data management practices in the success of parallelization strategies.

1.2. Objectives

The objectives for the research conducted in the HPGC activity that is documented in this report were to:

- Evaluate the applicability of cloud-optimized formats such as Cloud Optimized GeoTIFF (COG), GeoZarr, and GeoParquet to HPGC.
- Investigate the best practices for geospatial data indexing and partitioning, including adaptive indexing strategies and dynamic partitioning.
- Identify the optimal solution(s) for an HPC-optimized format.
- Quantify the performance improvements achieved through the adoption of optimized formats and data management strategies.

The study seeks to provide insights into future advancements in HPGC, focusing on geospatial data indexing and partitioning, which are critical to unlocking the full potential of parallelism in

HPGC and ensuring high-performance workflows can keep up with the ever-growing demands of the geospatial domain.

2

HPGC OPTIMIZED FORMATS

2 HPGC OPTIMIZED FORMATS

Three cloud optimized formats were analyzed and evaluated in Testbed-20: Cloud Optimized GeoTIFF (COG)[33][34], GeoZarr[30][31][32], and GeoParquet[36][35][1].

2.1. Benchmarking Considerations

The primary goal of benchmarking cloud-optimized formats in HPC is to evaluate their suitability in realizing their design benefits. Further detailed discussions on benchmarking design and related topics can be found in Annex B. The methodology to evaluate different cloud-optimized geospatial formats, such as COG, GeoZarr, and GeoParquet, in high-performance computing (HPC) environments involves a comprehensive benchmarking process. This process begins by assessing the impact of varying dataset sizes and complexities on performance. Evaluations were conducted across different storage environments, including object storage and traditional file systems, to determine how each format performs under various conditions. Typical access patterns, such as random reads, sequential reads, and writes, are analyzed to optimize data retrieval and storage efficiency. The suitability of each format for specific geospatial analyses is determined by testing various computational tasks. Additionally, different compression algorithms and levels are tested to evaluate their impact on storage efficiency and performance.

Benchmarking tests were designed to measure the time required to ingest large datasets and evaluate the performance of different compression methods. Random and sequential read performance was benchmarked to understand how efficiently each format handles data access. Common geospatial operations were tested to assess processing performance and scalability. The ability of each format to support parallel and distributed computing was evaluated to determine its effectiveness in HPC environments. Finally, the interoperability of each format with various software tools and libraries was examined, along with the ease of converting between cloud-optimized formats and traditional HPC data formats. This comprehensive evaluation provided insights into the strengths and limitations of each format, guiding the selection of the most suitable format for specific HPC applications.

2.2. HPC Optimized GeoTIFF Component

Cloud-Optimized GeoTIFF (COG) is designed to enhance the efficiency of streaming and partial downloading of web-based imagery and grid coverage data, facilitating fast visualization and processing. Unlike general GeoTIFFs, COGs group metadata together, allowing for the retrieval of all metadata in a single operation. Additionally, image data in COGs are organized in blocks, enabling internal chunking which supports efficient data access and manipulation.

A detailed description of benchmarks and their results can be found in Annex C. The benchmarking of COGs in high-performance computing (HPC) environments focused on

several key targets. Tile sizes of 256, 512, 1024, and 2048 pixels were tested alongside compression methods such as DEFLATE and LZW to evaluate their impact on performance. Read performance was compared between cloud storage and HPC storage to understand the efficiency of data retrieval. Processing performance was assessed by evaluating CPU and GPU utilization and scalability during tile processing tasks.

To test the COG format in HPC environments with real-world examples, a large set of datasets for aging dam assessment, collected and built by the I-GUIDE project[4], was used for in-depth experiments. The I-GUIDE HPC environment facilitated a series of experiments. The Common Workflow Language (CWL) was employed to create various deployable test workflows, and the Toil CWL runner [5] was used for workflow execution. GDAL was utilized for reading and writing COG files. A series of testing workflows and processes benchmarked and assessed the speed, scalability, and efficiency of COG in geospatial data processing workflows. The evaluation compared COG's performance with traditional GeoTIFF files.

The use of COGs in HPC environments offers several benefits. Efficient data access was achieved through internal compression, which supports partial reads and minimizes the need to decompress entire files. Optimized overviews, aligned with tiling components like 256×256 pixels, enhance data retrieval efficiency. Additionally, COGs provide flexible data management capabilities, allowing large datasets to be managed either as a single file or multiple smaller files based on specific requirements.

However, there are limitations to using COGs in HPC. Balancing between large files with extensive headers and numerous small files, which increase read requests, can affect performance. Smaller block sizes, such as 256×256 pixels, reduce unnecessary data reads but may increase the total number of read requests for large data portions. Furthermore, overviews are not directly applicable in HPC, and combining many small files for visualization can significantly impact load times, presenting challenges in efficiently managing and visualizing data.

2.3. HPC Optimized GeoZarr Component

GeoZarr is a cloud-optimized format designed for efficiently storing multidimensional array data, allowing for the storage of both vector and raster research data within a single structure. This versatility makes GeoZarr particularly useful for applications that require analyzing a series of images to detect changes over time, as it can synchronize imagery and features based on the time dimension. GeoZarr stores metadata in a single metadata file and organizes data in chunks, facilitating efficient data access and manipulation.

A detailed discussion of the benchmarking methodology, results, and discussions of GeoZarr in HPC can be found in Annex D. The benchmarking of GeoZarr in high-performance computing (HPC) environments focused on several key targets. Different chunk sizes and compression methods were tested to evaluate their impact on performance. The performance of parallel read/write operations was measured to assess how well GeoZarr supports parallel access. Processing efficiency was compared between cloud and HPC environments by evaluating processing times and resource utilization, including CPU and memory usage. These benchmarks

aimed to determine the speed, scalability, and efficiency of GeoZarr in handling complex geospatial data.

In this Testbed, the data collection for testing GeoZarr in HPC was the Hydroshare NetCDF data[4], which includes multidimensional time series data. The testing data selection criteria focused on the relevance of the imagery dataset, ensuring it referenced similar areas collected at various times. Evaluation Criteria assessed performance metrics like scalability and functionality, including data access, retrieval, management, updating, indexing, and partitioning. Tools and Techniques involved using open-source geospatial libraries and software, with I-GUIDE providing the HPC environment. Analysis methods included both qualitative analysis to document observations and comparison analysis to identify strengths and weaknesses. The experimentation phase involved staging collected imagery, installing software in a container, converting NetCDF data to a Zarr dataset, and using multiprocessing to process the data. The evaluation phase used a GitHub repository containing an API for writing GeoZarr datasets, with tests executed using the Python unittest module.

Using GeoZarr in HPC environments offers several benefits. Efficient data management is achieved through chunking, which supports scalable and efficient data processing. GeoZarr is designed for efficient parallel read/write operations, leveraging HPC capabilities to handle extensive datasets. Additionally, GeoZarr provides flexible storage options, as it can be stored in memory, on disk, in Zip files, and in cloud object storage such as S3, offering versatility in data management.

However, there are limitations to using GeoZarr in HPC. The format generates a large number of files, which can be problematic in traditional block-storage filesystems such as ext3/ext4, HFS +, and NTFS. Unexpected interactions, such as input/output (I/O) degradation, may occur when GeoZarr is used in block storage environments, which are common in HPC systems, instead of object storage environments like those in cloud computing[2]. Furthermore, GeoZarr is not always a one-size-fits-all solution, requiring careful consideration of the storage environment and potential adjustments for optimal performance. These limitations highlight the need for further research and development to fully optimize GeoZarr for HPC applications.

The OGC Testbed 20 participants recommend that coders applying GeoZarr in HPC environments be well-acquainted with both the data and the application use case, as data partitioning significantly impacts application performance. For larger datasets that will grow over time, it is crucial to keep partition sizes manageable to fit into the memory of processing nodes. Smaller partitions facilitate easier data insertion. Understanding the use case is essential because partitions can be created across all dataset dimensions. For instance, when reading raster data from a time-series, tiling the raster data in addition to partitioning by date can be beneficial. If the application requires reading subsets of the raster, tiling makes it quicker to access relevant parts. Coders should also recognize when distributing the creation and reading of a dataset is advantageous. Large datasets can benefit from partitioning to distribute processing tasks. Writing data from a stream, such as from an IoT device, can be optimized by feeding the writing process in a distributed manner, keeping partition sizes small and using locking methods to update dimension values serially.

In summary, GeoZarr is well-suited for processing on HPC clusters. Distributed programming and parallel processing support efficient writing and reading of large or streaming datasets. The main challenge for coders is determining the optimal data partitioning strategy. While large

partitions offer better compression rates, smaller partitions are more suitable for distributing processing across HPC cluster nodes.

Future investigations for effectively using GeoZarr in HPC could explore the new features introduced in Zarr version 3, which, despite being relatively new and not widely implemented, offer intriguing possibilities. Developing tools to leverage these features could be a focus of future work. Additionally, the OGC Testbed 20 participants did not have sufficient time to fully test distributed writing on HPC or to create code for reading from GeoZarr files. Future efforts could address these gaps by focusing on more extensive distributed testing and developing a reading API. Another area for future work is evaluating the Geo Extensions to Zarr, particularly those related to portrayal, which were not a focus of the Testbed task. Future investigations could explore how to create and deliver the portrayal aspects of GeoZarr files.

One notable feature of Zarr version 3 is the sharding index, which addresses the limitations imposed by storage devices and operating systems on the number and size of files. GeoZarr often deals with small chunk sizes, leading to numerous small files, which can be advantageous for distributing and modifying datasets in HPC environments but poses storage and system constraints. The sharding index allows a single file to hold multiple chunks of data, similar to how GeoTIFF files support internal tiling. This method enables better distribution of processing while reducing the complexity of adding and modifying data.

2.4. HPC Optimized GeoParquet Component

Cloud-optimized GeoParquet is a column-based file format designed to efficiently store and manage geospatial data. Unlike traditional row-oriented formats, GeoParquet organizes data into columns, enabling better compression and faster query performance. Each file is divided into row groups, with metadata stored at the end, facilitating easier writing processes. This structure makes GeoParquet particularly suitable for cloud environments, where efficient data access and storage are critical.

A detailed discussion of the benchmarking of GeoParquet in HPC can be found in Annex E. The experiments included testing reading and writing speed, cost of data storage and computing power, and parallel processing. Benchmarking GeoParquet in high-performance computing (HPC) environments focused on several key areas. The benefits of columnar storage for read-heavy workloads were assessed, as this format can significantly reduce the amount of data read from disk. Query performance was tested to measure the speed and efficiency of geospatial queries, which are crucial for many HPC applications. Additionally, data management strategies such as partitioning, bucketing, range groups, and paging were evaluated to determine their effectiveness in handling large datasets.

In this testbed, SpaceNet.ai data from challenge number 8 [6] [7], curated by Maxar Technologies for an area in Dernau, Germany, were used for two experiments to evaluate the performance of GeoZarr both qualitatively and comparatively. A set of open-source libraries and software supported the data creation, access, updating, and partitioning with GeoZarr. The experiments were conducted using the I-GUIDE HPC environment.

The use of GeoParquet in HPC offers several advantages. Its columnar layout results in smaller file sizes due to the compression of feature attributes, which is beneficial for storage efficiency.

GeoParquet supports random access to internal chunks, even when compressed, allowing for efficient data retrieval. Multithreading capabilities enable the replication of streaming downloads by fetching metadata first and then making multiple requests for internal chunks. Furthermore, GeoParquet's compatibility with existing tools and workflows enhances its interoperability, making it easier to integrate into current systems.

However, there are limitations to using GeoParquet in HPC. The placement of metadata at the end of the file can complicate the reading process, although this is generally manageable. The lack of a spatial index can reduce efficiency for certain applications that require quick access to specific spatial data. Additionally, partial data within a chunk cannot be fetched without decompressing the entire chunk, which can be inefficient. Handling large datasets often necessitates splitting them into multiple files, complicating data management and potentially impacting performance.

The experiments and investigations on GeoParquet in this Testbed concluded with the following recommendations for efficiently utilizing GeoParquet in an HPC environment:

- Containerization: While beneficial for keeping toolsets up to date and customizable, it complicates networked applications. Running applications directly on HPC hardware without containerization is recommended, though this may require code refactoring for compatibility with older software versions.
- Partitioning and Compression: Data curators should define page sizes that match the storage partitioning scheme and are large enough for effective compression. Smaller partitions allow better distribution access and less data refactoring, while larger partitions are better for static datasets with large query returns.
- Record Grouping: For spatial data, curators should group spatially near data together to improve query efficiency. Overlapping bounding boxes can be beneficial for large, spatially concentrated datasets to enable parallel processing.

The following summarizes some of the future directions for applications and advancement of GeoParquet for HPC:

- GeoParquet Features: Explore new features like indexing maps of data in a single column and handling complex, repeating data structures.
- Ray.io Framework[8]: Further investigate the use of Ray.io on HPC within container applications and explore other distributed frameworks that might work better.
- Open-Source Libraries: Continue improving open-source GeoParquet libraries by adding new features and enhancing distribution capabilities for HPC work. Focus on adding more support and resolving distribution issues to make these tools more valuable for HPC applications.

3

OUTLOOK

3 OUTLOOK

The future of High-Performance Geospatial Computing (HPGC) appears promising, with advancements in High Performance Computing (HPC) and cloud-optimized formats paving the way for more efficient and effective geospatial data processing and analysis. This Testbed-20 Report highlights the potential of integrating cloud-optimized formats within HPC environments to scale geospatial workflows more effectively, improve computational efficiency, and advance the use of parallelism in HPGC implementations.

Key research questions identified in this Report, including the potential use of existing cloud-optimized formats as HPC-optimized formats, best practices for geospatial data indexing, and partitioning, and optimal solutions for an HPC-optimized format, are expected to guide future investigations and developments in this field.

The OGC Testbed-20 HPGC activity evaluated cloud-optimized formats and investigated their applicability to HPGC. The results of this research could lead to the development of new HPC-optimized formats, further enhancing the capabilities of HPGC. Advancements in HPGC will likely focus on geospatial data indexing and partitioning, which are critical to unlocking the full potential of parallelism in HPGC and ensuring high-performance workflows can keep up with the ever-growing demands of the geospatial domain.

Future research should focus on further expanding the evaluation to a wider range of geospatial datasets and HPC platforms to assess the generalizability of the findings. Additionally, exploring the combination of different cloud-optimized formats for hybrid data models could be a promising avenue for further investigation.

As HPC capabilities continue to evolve, it becomes essential to develop adaptive indexing and partitioning strategies that can dynamically adjust to changing workload characteristics and data distributions.

As cloud-optimized formats evolve, their applications in HPC will also advance. Future investigations for GeoZarr could focus on leveraging new features in Zarr version 3, such as the sharding index, which helps manage file size limitations by allowing multiple chunks in a single file. This feature can improve data distribution and reduce the complexity of adding and modifying data. Additionally, more extensive testing of distributed writing and developing a reading API for GeoZarr are needed. Evaluating Geo Extensions to Zarr, particularly for portrayal aspects, could also be a valuable area of future work.

For GeoParquet, future directions include exploring new features like indexing maps of data in a single column and handling complex, repeating data structures. Investigating the use of Ray.io within HPC container applications and other distributed frameworks could enhance performance. Continuing to improve open-source GeoParquet libraries by adding new features and enhancing distribution capabilities will make these tools more valuable for HPC applications.

In addition to the cloud-optimized formats studied in this Testbed, other evolving formats need to be studied for their specialized applications and adaptations in HPC. Cloud-Optimized HDF/NetCDF[9] provides optimization for commonly used data formats for Earth observations. Cloud-Optimized Point Clouds (COPC)[10] are specifically designed for the LASER (LAS) file format to store 3-dimensional (x, y, z) point cloud data typically collected from LiDAR, where

an LAZ file is a compressed LAS file, and a COPC file is a valid LAZ file. FlatGeobuf[11] is another cloud-optimized binary file format for geographic vector data, such as points, lines, and polygons. PMTiles[12] is a cloud-optimized format for pyramids of map tiles in a single file on static storage. Different cloud-optimized formats may be adopted for specific domains or data, warranting further investigation and survey.

Furthermore, investigating the integration of machine learning techniques for optimizing data placement and access patterns within HPC environments could lead to substantial performance improvements. As generative AI and large model computations become increasingly prevalent, optimizing data formats for HPC environments is crucial. Here are some research directions to consider: enhanced data types and precision, distributed training and orchestration, sharding and partitioning techniques, integration with cloud-optimized formats, and performance benchmarking and testing.

4

SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

4 SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

The integration of High Performance Computing (HPC) in the geospatial sector, as discussed in this Report, introduces several security, privacy, and ethical considerations that require attention.

- Security: The application of HPC and cloud-optimized formats for geospatial data processing involves handling substantial amounts of data, some of which might be sensitive or confidential. Ensuring the security of this data is of utmost importance. Measures need implementation to protect data during transfer, processing, and storage, preventing unauthorized access and maintaining data integrity. This includes the application of secure data transfer protocols, encryption, and robust access control mechanisms.
- Privacy: Geospatial data often links to specific individuals or groups, raising significant privacy concerns. Ensuring compliance with relevant privacy laws and regulations, such as the European Union General Data Protection Regulation (GDPR), is essential. This includes obtaining necessary permissions, anonymizing data where possible, and using data only for its intended purpose.
- Ethical Considerations: The application of HPC for geospatial data processing has the potential to significantly impact various sectors, including meteorology, disaster management, and climate change. Considering the ethical implications of this work is crucial. This includes ensuring that the benefits of this technology are accessible to all, avoiding any potential harm, and considering the environmental impact of large-scale data processing.

While the integration of HPC and cloud-optimized formats in the geospatial sector offers significant benefits, addressing these security, privacy, and ethical considerations is crucial to ensure responsible and beneficial use of this technology. Future work in this area should continue to prioritize these considerations, developing robust solutions to these challenges as part of the ongoing advancement of High-Performance Geospatial Computing.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Apache Software Foundation: Apache Parquet, parquet

[2] Thomas Martin and Ward Fisher: netCDF vs Zarr, an Incomplete Comparison. University Corporation for Atmospheric Research (2024). https://www.unidata.ucar.edu/blogs/news/entry/netcdf-vs-zarr-an-incomplete

[3] G. Giacco, M. Manente, P. Gonçalves, M. Desruisseaux, and E. Rouault (eds.): OGC 21-032, OGC Testbed 17: COG/Zarr Evaluation Engineering Report. Open Geospatial Consortium (2022). https://docs.ogc.org/per/21-032.html

[4] Baig, F., J. Park: Inundation Maps Induced by Dam Failures — I-GUIDE Aging Dam Project, HydroShare (2024). http://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201

[5] toil-cwl-runner, https://github.com/DataBiosphere/toil

[6] Hänsch, R., Arndt, J., Lunga, D., Gibb, M., Pedelose, T., Boedihardjo, A., Petrie, D. and Bacastow, T.M.: Spacenet 8-the detection of flooded roads and buildings. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1472-1480 (2022).

[7] SN8: Flood Detection Challenge Using Multiclass Segmentation, https://spacenet.ai/sn8-challenge/

[8] Ray, https://www.ray.io/

[9] Aimee Barciauskas, Alexey Shikmonalov, and Luis Lopez: Cloud-Optimized HDF/NetCDF. Cloud Native Geospatial Foundation (2024). https://guide.cloudnativegeo.org/cloud-optimized-netcdf4-hdf5/

[10] copc.io, 2021, Cloud Optimized Point Cloud Specification – 1.0, https://copc.io/

[11] FlatGeobuf, https://flatgeobuf.org/

[12] PMTiles, https://github.com/protomaps/PMTiles

[13] I-GUIDE, https://i-guide.io/about-i-guide/

[14] Harinda Rajapaksha: A beginner's guide to SLURM. Oracle (2023) https://blogs.oracle.com/research/post/a-beginners-guide-to-slurm

[15] Official SLURM documentation, https://slurm.schedmd.com/

[16] SLURM job schedule, https://computing.fnal.gov/wilsoncluster/slurm-job-scheduler/

[17] Toil Documentation, https://toil.readthedocs.io

[18] GDAL, https://gdal.org

[19] Singularity, https://github.com/apptainer/singularity

[20] Apptainer, https://github.com/apptainer/apptainer

[21] Apptainer, https://apptainer.org/

[22] Apptainer Documentation, https://apptainer.org/documentation/

[23] RabbitMQ, https://www.rabbitmq.com

[24] RabbitMQ Server, https://github.com/rabbitmq/rabbitmq-server

[25] unittest — Unit testing framework, https://docs.python.org/3/library/unittest.html

[26] Michael Leedahl: geozarr, https://github.com/leedahl/geozarr

[27] Sharding codec (version 1.0), https://zarr-specs.readthedocs.io/en/latest/v3/codecs/sharding-indexed/v1.0.html

[28] Anyscale: Ray — The AI Compute Engine, https://www.ray.io/

[29] E. Yu and L. Di (eds.): OGC 23-044, OGC Testbed 19 High Performance Geospatial Computing Engineering Report. Open Geospatial Consortium (2024). https://docs.ogc.org/per/23-044.html

[30] zarr-developers: GeoZarr-spec, GeoZarr-spec

[31] Zarr Community: Zarr Storage Specification Version 2, zarr-v2

[32] Alistair Miles, Jonathan Striebel, Jeremy Maitin-Shepard (eds.): Zarr core specification Version 3.0. (2023), zarr-v3-core

[33] Joan Maso(ed.): OGC 21-026, OGC Cloud Optimized GeoTIFF Standard. Open Geospatial Consortium (2023). https://docs.ogc.org/is/21-026/21-026.html

[34] Cloud Optimized GeoTIFF, cog-web-page

[35] GeoParquet Specification, v1.1.0

[36] OGC: GeoParquet GitHub repository, geoparquet-github

A

ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

A ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

AOI	Area of Interest
API	Application Programming Interface
COG	Cloud Optimized GeoTIFF
COPC	Cloud-Optimized Point Cloud
CPU	Central Processing Unit
CWL	Common Workflow Language
DRU	Deploy, Replace, Undeploy
EOEPCA	Earth Observation Exploitation Platform Common Architecture
EPSG	Simple Linux Utility for Resource Management
GPU	Graphics Processing Unit
HPC	High-Performance Computing
HPGC	High-Performance Geospatial Computing
I-GUIDE	Institute for Geospatial Understanding through an Integrative Discovery Environment
IOT	Internet of Things
NSF	National Science Foundation
OGC	Open Geospatial Consortium
SLURM	Simple Linux Utility for Resource Management or Slurm Workload Manager
TIFF	Tagged Image File Format
UTM	Universal Transverse Mercator
WES	Workflow Execution Service

B

ANNEX B (NORMATIVE) BENCHMARKING CLOUD OPTIMIZED FORMATS IN HIGH PERFORMANCE COMPUTING ENVIRONMENTS

ANNEX B (NORMATIVE) BENCHMARKING CLOUD OPTIMIZED FORMATS IN HIGH PERFORMANCE COMPUTING ENVIRONMENTS

Benchmarking cloud-optimized formats, specifically Cloud Optimized GeoTIFF (COG), GeoZarr, and GeoParquet, in High Performance Computing (HPC) environments is crucial for understanding their performance and efficiency. These formats are designed to optimize storage and access patterns for geospatial data in cloud environments, but their performance in HPC settings requires thorough evaluation.

B.1. Benchmarking Considerations

When benchmarking these formats, several key factors should be considered: Data Size and Complexity: The size and complexity of the geospatial data will significantly impact performance. Larger datasets and more complex data structures may require different optimization strategies.

- Storage Environment: The type of storage environment (e.g., object storage, file system) can influence performance. Different formats may be better suited for specific storage types.
- Access Patterns: Understanding the typical access patterns (e.g., random reads, sequential reads, writes) is crucial for optimizing performance.
- Computational Tasks: The specific computational tasks that will be performed on the data will determine the most suitable format. Some formats may be better suited for certain types of analysis.
- Compression: Compression can significantly reduce storage requirements but may also impact performance. The choice of compression algorithm and compression level should be carefully considered

For the three cloud-optimized formats evaluated in Testbed-20, different key aspects may be highlighted.

B.1.1. Cloud Optimized GeoTIFF (COG)

COG products are pixel-interleaved GeoTIFF files with internal tiles that can be fetched individually. The files may include multiple reduced resolution pages for efficient viewing, though these are not relevant for full-resolution processing tasks. Key aspects of COG benchmarking include:

- Tile Size and Compression: Testing various tile sizes (256, 512, 1024, 2048 pixels) and compression algorithms (DEFLATE, LZW) to evaluate creation speed and compression efficiency.
- Read Performance: Assessing the time to read COGs from cloud vs. HPC storage, considering network effects and disk vs. object store efficiency.
- Processing Performance: Evaluating computational processing of tiles, including CPU/GPU utilization and scalability.

B.1.2. GeoZarr

GeoZarr is a format that stores geospatial data in chunked, compressed arrays. GeoZarr is designed for efficient cloud storage and access, with each chunk stored as a separate file. This supports better partitioning and parallel access, which can be advantageous in HPC environments.

- Chunking and Compression: Testing different chunk sizes and compression methods to determine the optimal configuration for both storage efficiency and read/write performance.
- Parallel Access: Evaluating the performance of parallel read/write operations, leveraging HPC's multiple hard drives and network capabilities.
- Processing Efficiency: Measuring the time and resource utilization for processing GeoZarr data, comparing cloud and HPC environments.

B.1.3. GeoParquet

GeoParquet[36] is an extension of the Apache Parquet format[1], optimized for geospatial data. The GeoParquet format supports efficient columnar storage, which can be beneficial for certain types of geospatial queries and analyses. Some benchmarking opportunities are:

- Columnar Storage: Assessing the performance benefits of columnar storage for geospatial data, particularly for read-heavy workloads.
- Query Performance: Testing the speed and efficiency of querying geospatial data stored in GeoParquet format.

- Data Management: GeoParquet supports different data management strategies, including:
  - Partitioning: Dividing data into distinct parts based on specific criteria (e.g., geographic regions) to improve query performance and manageability.
  - Bucketing: Grouping data into fixed-size buckets to optimize join operations and improve query efficiency.
  - Range Groups: Organizing data into ranges to facilitate efficient querying and data retrieval.
  - Paging: Implementing paging mechanisms to handle large datasets by retrieving data in smaller, manageable chunks.
- Compression and Encoding: Testing various compression and encoding schemes to find the best balance between storage efficiency and access speed.

B.2. Benchmarking Tests

Below are some potential benchmarking tests to evaluate the performance of cloud-optimized formats in an HPC environment:

- Data Ingestion:
  - Measure the time required to ingest large geospatial datasets into each format.
  - Evaluate the performance of different compression algorithms and compression levels.
- Data Access:
  - Benchmark random read and sequential read performance for different tile sizes and chunk sizes.
  - Measure the overhead of accessing data from cloud storage compared to local storage.
- Data Processing:
  - Test the performance of common geospatial operations (e.g., raster calculations, feature extraction) on data stored in each format.
  - Evaluate the scalability of processing tasks as the dataset size increases.
- Parallel Processing:
  - Assess the ability of each format to support parallel processing and distributed computing.

- Measure the performance of distributed algorithms on large datasets.

- Interoperability:
  - Evaluate the compatibility of each format with different software tools and libraries.
  - Test the ability to convert between different formats.

B.2.1. COG

The following are some specific benchmarking tests for COG in HPC:

1. Test the ability to create an uncompressed COG with tile sizes of 256, 512, 1024, and 2048 pixels.
2. Test the ability to create a DEFLATE compressed COG with tile sizes of 256, 512, 1024, and 2048 pixels.
3. Test the ability to create an LZW compressed COG with tile sizes of 256, 512, 1024, and 2048 pixels.
4. Test the ability to sequentially read and download a complete COG with tile sizes of 256, 512, 1024, and 2048 pixels.
5. Test the ability to randomly read 10,000 tiles from 10 processes for a COG with tile sizes of 256, 512, 1024, and 2048 pixels.
6. Test the ability to process a COG with tile sizes of 256, 512, 1024, and 2048 pixels.

C

ANNEX C (NORMATIVE) HPC OPTIMIZED GEOTIFF COMPONENT

ANNEX C (NORMATIVE) HPC OPTIMIZED GEOTIFF COMPONENT

C.1. Introduction

The COG format is a cloud-optimized version of the GeoTIFF format that enables efficient access to geospatial data stored in the cloud. By structuring the GeoTIFF data in a way that supports partial reads and writes, COG files can be accessed and processed more efficiently.

C.2. Methodology

This section outlines the methodology for evaluating the capabilities of COG within High Performance Geospatial Computing (HPGC) environments. The evaluation process included data collection, selection criteria, evaluation criteria, tools and techniques, and analysis methods. The evaluation examined a use case for processing large-scale geospatial data stored in COG format.

C.2.1. Data Collection

1. **Data Sources:** Use the GeoTIFF files provided by I-GUIDE [13] and accessible through https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/ [4]. I-GUIDE is the National Science Foundation (NSF) Institute for Geospatial Understanding through an Integrative Discovery Environment [13].

C.2.2. Evaluation Criteria

C.2.2.1. Performance Metrics:

- **Speed:** Measure data read/write speeds for accessing and processing COG files.

- **Scalability:** Assess how well COG files handle increasing data volume and complexity.
- **Efficiency:** Evaluate computational efficiency in terms of resource usage (CPU, memory).

C.2.2.2. Functionality:

- **Data Access:** Test the ease of data access and retrieval from COG files.
- **Data Management:** Evaluate capabilities for data updating, indexing, and partitioning within COG files.

C.2.3. Tools and Techniques

1. **Geospatial Tools:** Use open-source geospatial libraries and software to work with COG files.
2. **Benchmarking Tools:** Collect CPU and memory resource utilization metrics and timing metrics for performance evaluation.
3. **HPC Environment:** Utilize the High-Performance Computing environment provided by I-GUIDE for this investigation.

C.2.4. Analysis Methods

1. **Quantitative Analysis:**
   - Perform statistical analysis on collected performance data.
   - Use visualization tools to create performance graphs and charts.
   - Compare COG's performance against a variety of data types and processing scenarios.
2. **Qualitative Analysis:**
   - Document observations regarding functionality and ease of use of COG files.
   - Examine the benefits and drawbacks of using COG files in an HPC environment.
3. **Comparison Analysis:**
   - Compare the results of processing geospatial data stored in COG format with traditional GeoTIFF files in terms of speed, scalability, efficiency, and functionality.

C.3. Experimentation

The following sub-sections describe the experiments conducted, including the setup, process, and results.

C.3.1. Workflows handling COG files

This section describes the setup of the High-Performance Computing (HPC) environment for the evaluation of COG files. This environment is relevant for other cloud-optimized geospatial data formats covered by this report.

To encapsulate the workflow, the Testbed participants used the Common Workflow Language (CWL), and to run on HPC they relied on the Toil CWL runner [5]. The CWL is a specification for describing analysis workflows and tools in a way that makes them portable and scalable across a variety of software and hardware environments. Toil is a Python-based workflow engine that can execute CWL workflows on a variety of execution environments, including HPC clusters and especially SLURM [16] [14] [15] which is the available scheduler on the HPC the Testbed used for these experiments.

The Toil CWL runner [5] [17] was installed on the HPC environment provided by I-GUIDE for this investigation.

```
python3 -m virtualenv ~/venv
source ~/venv/bin/activate
pip install "toil[cwl]"
```

Listing C.1 — Install Toil CWL runner

Since GDAL[18] supports COG, GDAL was used to read and write COG files. The GDAL library is a widely used open-source library for reading and writing geospatial data formats.

The code snippet below shows the use of `gdal_translate` in a CWL workflow that reads a COG file, performs a geographic subset, and writes the result to a new COG file.

The CWL file defines the command-line tool `gdal_translate` with input parameters for the bounding box, EPSG code, and asset URL. The tool uses inline JavaScript expressions to construct the command-line arguments dynamically based on the input parameters.

```
class: CommandLineTool

requirements:
  InlineJavascriptRequirement: {}
  DockerRequirement:
    dockerPull: osgeo/gdal:alpine-normal-3.6.2

baseCommand: gdal_translate
arguments:
- -projwin
- valueFrom: ${ return inputs.bbox.split(",")[0]; }
- valueFrom: ${ return inputs.bbox.split(",")[3]; }
```

```
- valueFrom: ${ return inputs.bbox.split(",")[2]; }
- valueFrom: ${ return inputs.bbox.split(",")[1]; }
- -projwin_srs
- valueFrom: ${ return inputs.epsg; }
- valueFrom: |
    ${ if (inputs.asset.startsWith("http")) {

          return "/vsicurl/" + inputs.asset;

       } else {

          return inputs.asset;

       }
    }
- valueFrom: ${ return inputs.asset.split("/").slice(-1)[0]; }

inputs:
  compress:
    type: string
  blocksize:
    type: int
  asset:
    type: string
  bbox:
    type: string
  epsg:
    type: string
    default: "EPSG:4326"

outputs:
  tifs:
    outputBinding:
      glob: '*.tif'
    type: File

cwlVersion: v1.0
```

Listing C.2 — Example CWL file for `gdal_translate`

The CWL file defines the input parameters for the `gdal_translate` tool, including the asset URL, bounding box, and EPSG code. To set values to these input parameters, a YAML file is used.

```
bbox: "136.659,-35.96,136.923,-35.791"
asset: https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/53/
H/PA/2021/7/S2B_53HPA_20210723_0_L2A/B8A.tif
epsg: "EPSG:4326"
```

Listing C.3 — Example YAML file for `gdal_translate` inputs parameters

Toil CWL runner [5] [17] was used to execute the workflow defined in the CWL file. The runner is configured to use the SLURM batch system [16], disable caching, and run the workflow in a Singularity (renamed Apptainer) container [22] [20] [21] [19].

```
TOIL_CHECK_ENV=True toil-cwl-runner \
        --batchSystem slurm \
        --disableCaching true \
        --workDir . \
```

```
        --jobStore ./store_stats \
        --clean=never \
        --singularity \
        example.cwl example-job.yaml
```

Listing C.4 — Execute `gdal_translate` through Toil CWL runner

The execution of the workflow through toil-cwl-runner [5] generated the following debug messages on the standard output.

```
[2024-11-01T06:04:04-0400] [MainThread] [I] [cwltool] Resolved 'example.cwl' to
'file:///home/x-gfenoy/TB20-Benchmark/gdal_translate/gdt1/example.cwl'
[2024-11-01T06:04:05-0400] [MainThread] [I] [toil.job] Saving graph of 1 jobs, 1
new
[2024-11-01T06:04:05-0400] [MainThread] [I] [toil.job] Processing job 'CWLJob'
gdal_translate kind-CWLJob/instance-wm_sphxd v0
[2024-11-01T06:04:05-0400] [MainThread] [I] [toil] Running Toil version 5.5.0-b0f
f5be051f2fd55352e00450b7848dcf8354a3b on host login07.anvil.rcac.purdue.edu.
[2024-11-01T06:04:05-0400] [MainThread] [I] [toil.leader] Issued job 'CWLJob'
gdal_translate kind-CWLJob/instance-wm_sphxd v1 with job batch system ID: 0 and
cores: 1, disk: 3.0 Gi, and memory: 2.0 Gi
[2024-11-01T06:04:07-0400] [MainThread] [I] [toil.leader] 0 jobs are running, 1
jobs are issued and waiting to run
[2024-11-01T06:05:55-0400] [MainThread] [I] [toil.leader] Finished toil run
successfully.

Workflow Progress 100%|
                                                                              | 1/1 (0
failures) [01:12<00:00, 0.01 jobs/s]
{
    "tifs": {
        "location": "file:///home/x-gfenoy/TB20-Benchmark/gdal_translate/gdt1/
B8A.tif",
        "basename": "B8A.tif",
        "nameroot": "B8A",
        "nameext": ".tif",
        "class": "File",
        "checksum": "sha1$679b939185992198bfd2d0ed15f1178fc661472a",
        "size": 2275155
    }
}[2024-11-01T06:05:56-0400] [MainThread] [I] [toil.common] Successfully deleted
the job store: FileJobStore(/home/x-gfenoy/TB20-Benchmark/gdal_translate/gdt1/
store_stats)
```

Listing C.5 — Output generated by the execution

Toil[17] comes with a `stats` command that can be used to generate statistics about the execution of the workflow. This command provides information about 4 categories: memory, wait, time, and clock. The following command generates statistics about memory usage.

```
toil stats store_stats --categories memory --human
```

Listing C.6 — Generate statistics with Toil `stats`

The `stats` command generates detailed information about the execution of the workflow.

```
Batch System: slurm
Default Cores: 1  Default Memory: 2.0G
```

```
Max Cores: 9.22337e+18
Total Clock: 2s  Total Runtime: 74s
Worker
    Count |                                    Memory
        n |       min     med     ave     max   total
        1 |        0K      0K      0K      0K      0K
Job
 Worker Jobs  |     min    med    ave    max
              |       0      0      0      0
    Count |                                    Memory
        n |       min     med     ave     max   total
        0 |        0K      0K      0K      0K      0K
```

Listing C.7 — Output generated by the `stats` command

This example run was not intended to be representative of a real-world testing scenario, but rather to illustrate the process of executing a CWL workflow on an HPC environment using Toil[17].

In the previous example the `gdal_translate` tool was used to read COG files. The same approach can be used to write COG files with GDAL.

C.3.2. Toil Workflow Execution Service (WES)

The Toil Workflow Execution Service (WES) is a RESTful API that provides a standardized interface for executing CWL workflows on a variety of execution environments. The WES API allows users to submit, monitor, and retrieve the status of CWL workflows.

To use the WES API, the Toil WES server must be installed and configured on the HPC environment. Toil WES requires a RabbitMQ server [23] [24] that can be started with the following command:

```
apptainer run \
    --bind $(pwd)/etc:/opt/bitnami/rabbitmq/etc/rabbitmq/ \
    --bind $(pwd)/var_lib:/opt/bitnami/rabbitmq/var/lib/rabbitmq \
    --bind $(pwd)/.rabbitmq:/opt/bitnami/rabbitmq/.rabbitmq/ \
    --bind $(pwd)/tmp:/bitnami/rabbitmq/ \
    docker://bitnami/rabbitmq:latest
```

Listing C.8 — Start the RabbitMQ server

The `toil.server.celery_app` can be found in the file venv3.11/lib/python3.11/site-packages/toil/server/celery_app.py. If the environment variable `TOIL_WES_BROKER_URL` is defined, it is the one used to access the RabbitMQ server (per default the user can use `user` as username and `bitnami` as password).

```
source ~/venv3.11/bin/activate
TOIL_WES_BROKER_URL="amqp://user:bitnami@127.0.0.1:5672" celery -A toil.server.
celery_app worker --loglevel=INFO
```

Listing C.9 — Start Celery workers

The WES server can be started with the following command:

```
source ~/venv3.11/bin/activate
TOIL_WES_BROKER_URL=amqp://user:bitnami@127.0.0.1:5672//  toil server \
  --opt=--stats \
```

```
  --opt=--batchSystem=slurm \
  --opt=--defaultMemory=2Gi \
  --opt=--maxMemory=100Gi \
  --opt=--singularity \
  --opt=--jobStore=/home/x-gfenoy/TOIL/toil_storage/example/job_store/$(uuidgen)
```

Listing C.10 — Start the Toil WES server

From there the WES API can be accessed through the URL: http://localhost:8080/ga4gh/wes/v1/service-info and provide information about the service.


```
{
  "version": "7.0.0",
  "workflow_type_versions": {
    "py": {
      "workflow_type_version": [
        "3.7",
        "3.8",
        "3.9"
      ]
    },
    "cwl": {
      "workflow_type_version": [
        "v1.0",
        "v1.1",
        "v1.2"
      ]
    },
    "wdl": {
      "workflow_type_version": [
        "draft-2",
        "1.0"
      ]
    }
  },
  "supported_wes_versions": [
    "1.0.0"
  ],
  "supported_filesystem_protocols": [
    "file",
    "http",
    "https"
  ],
  "workflow_engine_versions": {
    "toil": "7.0.0"
  },
  "default_workflow_engine_parameters": [],
  "system_state_counts": {
    "SYSTEM_ERROR": 2
  },
  "tags": {
    "wes_dialect": "standard"
  }
}
```


Listing C.11 — WES service-info URL

By using the Toil WES API, client applications can submit CWL workflows to the HPC environment for execution and monitor their progress through the Toil WES API.

The client application used for this benchmark was ZOO-Project with support for Deploy, Replace, Undeploy (DRU) and CWL, a reference implementation of the OGC API — Processes

— Part 1: Core Standard. ZOO-Project provides a web processing service that can execute CWL workflows in a HPC environment. In the Earth Observation Exploitation Platform Common Architecture (EOEPCA) project, an implementation of the zoo-wes-runner and its associated cookiecutter template was used to execute CWL workflows on the HPC environment using the Toil WES.

To enable access from the ZOO-Project Kubernetes cluster to the Toil WES server, SSH tunneling was used to forward the port 8080 from the HPC environment to port 8100 on the local machine.

```
ssh -L 0.0.0.0:8100:127.0.0.1:8080 username@hpc-hostname
```

Listing C.12 — SSH tunneling to forward the port 8080

To enable access from the HPC environment to the ZOO-Project Kubernetes cluster, SSH tunneling was used to forward the port 4566 from the local machine to the port 4900 on the HPC environment.

```
ssh -R 127.0.0.1:4900:127.0.0.1:4566 username@hpc-hostname
```

Listing C.13 — SSH tunneling to forward the port 4566

The ZOO-Project instance can then be configured to use the Toil WES API to execute CWL workflows on the HPC environment.

```
workflow.inputs.WES_URL: "http://<WES-HOSTNAME>>:8100/ga4gh/wes/v1/"
workflow.inputs.WES_USER: "test"
workflow.inputs.WES_PASSWORD: "$$2y$$12$$ci.4U63YX83CwkyUrjqxAucnmi2xXOIlEF6T/
KdP9824f1Rf1iyNG"
```

Listing C.14 — ZOO-Project configuration

To deploy the ZOO-Project-DRU instance with the WES support on a local Kubernetes cluster, the following command can be used:

```
git clone https://github.com/ZOO-Project/ZOO-Project.git
cd ZOO-Project
skaffold dev -p dru-wes
```

Listing C.15 — Deploy the ZOO-Project-DRU with WES support on a local cluster

The ZOO-Project-DRU instance with WES support can then be accessed through the URL: http://localhost:8080/.

The ZOO-Project-DRU instance provides an API implementation compliant with the OGC API — Processes — Part 1: Core Standard, that can execute CWL workflows on the HPC environment using the Toil WES API. In addition to the execution capability, the ZOO-Project-DRU provides also an API implementing the OGC API — Processes: Part 2: Deploy, Replace, Undeploy candidate Standard to manage the deployment of CWL workflows.

C.3.3. Vulnerability Analysis for Aging Dams Workflow

The workflow to be supported is available here: https://github.com/I-GUIDE/vulnerability_analysis_agingdams and was proposed by the I-GUIDE project.

The workflow is available as a Jupyter notebook: https://github.com/I-GUIDE/vulnerability_analysis_agingdams/blob/main/iguide_agingdams.ipynb.

The workflow is composed of several steps that are able to be converted into CWL.

Nevertheless, the ability to read the data from some external servers was required, but was not possible due to restricted access.

For example, for Step 0 described in the notebook, the data is read from the following server: https://fim.sec.usace.army.mil. However, this URL is not accessible.

As the required data were available on the HPC environment, these data could be used for testing purposes to replicate Step 1 of the workflow. Step 1 is titled Extract "Inundation Classification for Census Tracts for this Dam".

The part of the notebook that is of interest is the following:

```
import cybergis_compute_client
from cybergis_compute_client import CyberGISCompute

cybergis = CyberGISCompute(url="cgjobsup.cigi.illinois.edu", isJupyter=True,
protocol="HTTPS", port=443, suffix="v2")
cybergis.show_ui(defaultJob="Extract_Inundation_Census_Tracts_Processor",input_
params=params_classify_tracts)
```

Listing C.16 — I-GUIDE CyberGISCompute invocation

The `cybergis_compute_client` is a Python client library that provides an interface to the CyberGISCompute API. The `CyberGISCompute` class is used to create a connection to the CyberGISCompute server and submit a job to the server.

In order to replicate this step using a CWL workflow, a Docker image for the Job named "Extract_Inundation_Census_Tracts_Processor" needs to be created. For doing so, the following repository can be used: https://github.com/I-GUIDE/container_images/. The I-GUIDE project uses the HPC Container Maker from NVIDIA to build the Docker or Apptainer images.

For converting the recipe into a Dockerfile, use the following command.

```
hpccm --recipe recipe.hpccm --format docker > Dockerfile
```

Listing C.17 — Use HPC Container Maker to convert a recipe to a Dockerfile

See below the slightly modified generated Dockerfile.

```
FROM geoedf/framework

COPY ./GeoEDF /extractinundationcensustracts/GeoEDF
COPY ./setup.py /extractinundationcensustracts/setup.py

RUN apt-get update && \
    apt-get install -y unzip

RUN cd /extractinundationcensustracts && \
    pip3 install .

RUN echo ". /opt/conda/etc/profile.d/conda.sh" >> ~/.bashrc && \
    echo "conda activate geoedf" >> ~/.bashrc
```

```
ENV PATH="/opt/conda/envs/geoedf/bin:${PATH}"
COPY ./main_multi_scenarios.py /app.py
```

Listing C.18 — Generated Dockerfile

To test the container for executing the job, follow the steps from the `manifest.json` available from: https://github.com/cybergis/population_vulnerable_to_dam_failure_compute. The steps are as follows: `pre_processing_stage`, `execution_stage`, `post_processing_stage`.

```
python setup.py
python main_multi_scenarios_final_JP.py
python cleanup.py
```

Listing C.19 — Steps defined in the manifest.json

A CWL encapsulating the workflow is provided. Execute the CWL using `cwltool`.

```
# cwltool --leave-tmpdir --tmpdir-prefix=$(pwd)/ ../app-pkg.cwl#extrac-
inundation-census-tract --dam_id_array="CA10022"
INFO /home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-inundation-census-tract-
app-pkg/venv/bin/cwltool 3.1.20250110105449
INFO Resolved '../../app-pkg.cwl#extrac-inundation-census-tract' to 'file:///
home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-inundation-census-tract-app-
pkg/app-pkg.cwl#extrac-inundation-census-tract'
INFO [workflow ] start
INFO [workflow ] starting step step_1
INFO [step step_1] start
WARNING [job step_1] Skipping Docker software container '--memory' limit despite
presence of ResourceRequirement with ramMin and/or ramMax setting. Consider
running with --strict-memory-limit for increased portability assurance.
WARNING [job step_1] Skipping Docker software container '--cpus' limit despite
presence of ResourceRequirement with coresMin and/or coresMax setting. Consider
running with --strict-cpu-limit for increased portability assurance.
INFO [job step_1] /home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-inundation-
census-tract-app-pkg/runs/tmp1/uzihaaj0$ docker \
    run \
    -i \
    --mount=type=bind,source=/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-
inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0,target=/LtLiFg \
    --mount=type=bind,source=/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-
inundation-census-tract-app-pkg/runs/tmp1/z7d58kky,target=/tmp \
    --workdir=/LtLiFg \
    --read-only=true \
    --user=1001:1001 \
    --rm \
    --cidfile=/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-inundation-
census-tract-app-pkg/runs/tmp1/f92bqcma/20250131164539-392399.cid \
    --env=TMPDIR=/tmp \
    --env=HOME=/LtLiFg \
    geoedf/extractinundationcensustracts4cwl \
    /bin/bash \
    run.sh \
    CA10022 2> /home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-inundation-
census-tract-app-pkg/runs/tmp1/uzihaaj0/std.out
/LtLiFg
Output Directory:  /tmp/Multi_F_Results/N_0
Temp Directory:  /LtLiFg
Total Dams: 345
Creating output file that is 197P x 137L.
```

```
Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.org/
resource/1de14e8622564aa5bd645b92d899a201/data/contents//MH_F_CA10022.tiff [1/
1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=no&empty_
dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/
data/contents//MH_F_CA10022.tiff.
Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=https://
www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//MH_F_
CA10022.tiff to destination /LtLiFg/MH_F_CA10022_resample.tiff.
...10...20...30...40...50...60...70...80...90...100 - done.
0...10...20...30...40...50...60...70...80...90...Several drivers matching json
extension. Using GeoJSON
Creating output /LtLiFg/MH_F_CA10022.json of format GeoJSON.
100 - done.
Creating output file that is 190P x 129L.
Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.org/
resource/1de14e8622564aa5bd645b92d899a201/data/contents//TAS_F_CA10022.tiff [1/
1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=no&empty_
dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/
data/contents//TAS_F_CA10022.tiff.
Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=https://
www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//TAS_
F_CA10022.tiff to destination /LtLiFg/TAS_F_CA10022_resample.tiff.
...10...20...30...40...50...60...70...80...90...100 - done.
0...10...20...30...40...50...60...70...80...90...Several drivers matching json
extension. Using GeoJSON
Creating output /LtLiFg/TAS_F_CA10022.json of format GeoJSON.
100 - done.
Creating output file that is 148P x 103L.
Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.org/
resource/1de14e8622564aa5bd645b92d899a201/data/contents//NH_F_CA10022.tiff [1/
1] : 0Using internal nodata values (e.g. -9999) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//NH_F_CA10022.tiff.
Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=https://
www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//NH_F_
CA10022.tiff to destination /LtLiFg/NH_F_CA10022_resample.tiff.
...10...20...30...40...50...60...70...80...90...100 - done.
0...10...20...30...40...50...60...70...80...90...Several drivers matching json
extension. Using GeoJSON
Creating output /LtLiFg/NH_F_CA10022.json of format GeoJSON.
100 - done.
Archive:  /tmp/tl_2020_06_tabblock20.zip
 extracting: tl_2020_06_tabblock20.cpg
  inflating: tl_2020_06_tabblock20.dbf
  inflating: tl_2020_06_tabblock20.prj
  inflating: tl_2020_06_tabblock20.shp
  inflating: tl_2020_06_tabblock20.shp.ea.iso.xml
  inflating: tl_2020_06_tabblock20.shp.iso.xml
  inflating: tl_2020_06_tabblock20.shx
CA10022: Step 1, 1/4, Identifying associated regions for multiple scenarios
CA10022: Step 1, 2/4, Search states associated
-- CA10022 impacts 1 States, ['06']
CA10022: Step 1, 3/4, Extracting GEOID of census blocks
CA10022: Step 1, 4/4, Assigning geometry to census blocks
CA10022: Step 2, 1/2, Merging census data to FIM geoid
CA10022: Step 2, 2/2, Calculating Moran's I and LISA
Merging results for 1 dams
Saving results Multi_F_fim.geojson
Saving results Multi_F_mi.geojson
Saving results Multi_F_lm.geojson
Done
INFO [job step_1] Max memory used: 6980MiB
INFO [job step_1] completed success
```

```
INFO [step step_1] completed success
INFO [workflow ] completed success
{
    "stac": [
        {
            "location": "file:///home/gerald-fenoy/WORK/2024/TB20/cybergis-
extract-inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0",
            "basename": "uzihaaj0",
            "class": "Directory",
            "listing": [
                {
                    "class": "File",
                    "location": "file:///home/gerald-fenoy/WORK/2024/TB20/
cybergis-extract-inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0/Multi_F_mi.
geojson",
                    "basename": "Multi_F_mi.geojson",
                    "size": 1510,
                    "checksum": "sha1$99fa0fdb37b17ad7cdbd10159acc24f5e3ec0a0b",
                    "path": "/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-
inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0/Multi_F_mi.geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/gerald-fenoy/WORK/2024/TB20/
cybergis-extract-inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0/Multi_F_lm.
geojson",
                    "basename": "Multi_F_lm.geojson",
                    "size": 2019858,
                    "checksum": "sha1$455d6d5d6c54d58c9d2d817704978ea445c4d2a2",
                    "path": "/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-
inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0/Multi_F_lm.geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/gerald-fenoy/WORK/2024/TB20/
cybergis-extract-inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0/Multi_F_fim.
geojson",
                    "basename": "Multi_F_fim.geojson",
                    "size": 17870,
                    "checksum": "sha1$228acbc5fb546fc448cbbf881c0b621b62de8ea3",
                    "path": "/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-
inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0/Multi_F_fim.geojson"
                }
            ],
            "path": "/home/gerald-fenoy/WORK/2024/TB20/cybergis-extract-
inundation-census-tract-app-pkg/runs/tmp1/uzihaaj0"
        }
    ]
}INFO Final process status is success
```

Listing C.20 — Run log from `cwltool`

Below is the log generated by the execution of the workflow using the `toil-cwl-runner`.

```
TOIL_CHECK_ENV=True toil-cwl-runner --jobStore=/home/x-gfenoy/TOIL/toil_storage/
example/job_store/$(uuidgen) --batchSystem=slurm         --singularity
--defaultDisk=6G --maxDisk=8G app-package.cwl#extrac-inundation-census-tract
example-job.yaml
[2025-02-01T06:21:48-0500] [MainThread] [I] [cwltool] Resolved 'app-package.
cwl#extrac-inundation-census-tract' to 'file:///home/x-gfenoy/cybergis/app-
package.cwl#extrac-inundation-census-tract'
[2025-02-01T06:21:49-0500] [MainThread] [I] [toil.cwl.cwltoil] Importing input
files...
```

```
[2025-02-01T06:21:49-0500] [MainThread] [I] [toil.cwl.cwltoil] Importing tool-
associated files...
[2025-02-01T06:21:49-0500] [MainThread] [I] [toil.cwl.cwltoil] Creating root job
[2025-02-01T06:21:49-0500] [MainThread] [I] [toil.cwl.cwltoil] Starting workflow
[2025-02-01T06:21:49-0500] [MainThread] [I] [toil] Running Toil version 7.0.0-d56
9ea5711eb310ffd5703803f7250ebf7c19576 on host login07.anvil.rcac.purdue.edu.
[2025-02-01T06:21:51-0500] [MainThread] [I] [toil.leader] 0 jobs are running, 1
jobs are issued and waiting to run
[2025-02-01T06:21:51-0500] [MainThread] [I] [toil.leader] Issued job 'CWLJob'
extrac-inundation-census-tract.0.process kind-CWLJob/instance-0udrq2p2 v1 with
job batch system ID: 2 and disk: 5.9 Gi, memory: 7.9 Gi, cores: 1, accelerators:
[], preemptible: False
[2025-02-01T06:23:51-0500] [Thread-4 (statsAndLoggingAggregator)] [I] [toil.
statsAndLogging] Got message from job at time 02-01-2025 06:23:51: CWL step
complete: extrac-inundation-census-tract.0.process
[2025-02-01T06:23:51-0500] [Thread-4 (statsAndLoggingAggregator)] [I] [toil.
statsAndLogging] extrac-inundation-census-tract.0.process.stdout follows:
=========>
        /zaPGJB
        Output Directory:  /tmp/Multi_F_Results/N_0
        Temp Directory:  /zaPGJB
        Total Dams: 345
        Creating output file that is 197P x 137L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//MH_F_CA10022.tiff
[1/1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//MH_F_CA10022.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//MH_F_CA10022.tiff to destination /zaPGJB/MH_F_CA10022_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /zaPGJB/MH_F_CA10022.json of format GeoJSON.
        100 - done.
        Creating output file that is 190P x 129L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//TAS_F_CA10022.
tiff [1/1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//TAS_F_CA10022.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//TAS_F_CA10022.tiff to destination /zaPGJB/TAS_F_CA10022_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /zaPGJB/TAS_F_CA10022.json of format GeoJSON.
        100 - done.
        Creating output file that is 148P x 103L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//NH_F_CA10022.tiff
[1/1] : 0Using internal nodata values (e.g. -9999) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//NH_F_CA10022.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//NH_F_CA10022.tiff to destination /zaPGJB/NH_F_CA10022_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /zaPGJB/NH_F_CA10022.json of format GeoJSON.
```

```
        100 - done.
        Archive:  /tmp/tl_2020_06_tabblock20.zip
         extracting: tl_2020_06_tabblock20.cpg
          inflating: tl_2020_06_tabblock20.dbf
          inflating: tl_2020_06_tabblock20.prj
          inflating: tl_2020_06_tabblock20.shp
          inflating: tl_2020_06_tabblock20.shp.ea.iso.xml
          inflating: tl_2020_06_tabblock20.shp.iso.xml
          inflating: tl_2020_06_tabblock20.shx
        CA10022: Step 1, 1/4, Identifying associated regions for multiple
scenarios
        CA10022: Step 1, 2/4, Search states associated
        -- CA10022 impacts 1 States, ['06']
        CA10022: Step 1, 3/4, Extracting GEOID of census blocks
        CA10022: Step 1, 4/4, Assigning geometry to census blocks
        CA10022: Step 2, 1/2, Merging census data to FIM geoid
        CA10022: Step 2, 2/2, Calculating Moran's I and LISA
        Merging results for 1 dams
        Saving results Multi_F_fim.geojson
        Saving results Multi_F_mi.geojson
        Saving results Multi_F_lm.geojson
        Done
<=========
[2025-02-01T06:24:04-0500] [MainThread] [I] [toil.leader] Finished toil run
successfully.

Workflow Progress 100%|
                                                                          | 4/4 (0
failures) [02:09<00:00, 0.03 jobs/s]
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Collecting
workflow outputs...
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-out518lljks/Multi_F_fim.geojson...
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-out518lljks/Multi_F_mi.geojson...
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-out518lljks/Multi_F_lm.geojson...
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Stored workflow
outputs
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Computing output
file checksums...
{
    "stac": [
        {
            "location": "file:///home/x-gfenoy/cybergis/tmp-out518lljks",
            "basename": "tmp-out518lljks",
            "nameroot": "tmp-out518lljks",
            "nameext": "",
            "class": "Directory",
            "listing": [
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-out518lljks/
Multi_F_fim.geojson",
                    "basename": "Multi_F_fim.geojson",
                    "size": 17870,
                    "checksum": "sha1$228acbc5fb546fc448cbbf881c0b621b62de8ea3",
                    "nameroot": "Multi_F_fim",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-out518lljks/Multi_F_fim.
geojson"
```

```
                },
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-out518lljks/
Multi_F_mi.geojson",
                    "basename": "Multi_F_mi.geojson",
                    "size": 1514,
                    "checksum": "sha1$e5c8788ddd4e44dd59c1ebc72208a97f485b5af4",
                    "nameroot": "Multi_F_mi",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-out518lljks/Multi_F_mi.
geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-out518lljks/
Multi_F_lm.geojson",
                    "basename": "Multi_F_lm.geojson",
                    "size": 2019858,
                    "checksum": "sha1$455d6d5d6c54d58c9d2d817704978ea445c4d2a2",
                    "nameroot": "Multi_F_lm",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-out518lljks/Multi_F_lm.
geojson"
                }
            ],
            "path": "/home/x-gfenoy/cybergis/tmp-out518lljks"
        }
    ]
}
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.cwl.cwltoil] CWL run complete!
[2025-02-01T06:24:05-0500] [MainThread] [I] [toil.common] Successfully deleted
the job store: FileJobStore(/home/x-gfenoy/TOIL/toil_storage/example/job_store/
445e5d39-99a0-4b17-b4ac-5fb9e05efdd6)
```

Listing C.21 — Run log from toil-cwl-runner

In order to execute the workflow via toil using SLURM as a batch system, publish the following docker image on DockerHub: geolabs/cybergis-eoap.

The scatter mechanism offered by CWL was used and is described in the following section as to how to process multiple dams in parallel.

C.3.4. Scatter mechanism in CWL

The CWL scatter mechanism supports parallel execution of workflow steps, where each scattered execution instance receives a different subset of the input data. This improves performance and resource utilization, especially when dealing with large-scale datasets. CWL provides different scattering methods:

- dot_product — Executes workflow steps in parallel by pairing corresponding elements from multiple input arrays.
- flat_crossproduct — Combines every element from one input array with every element of another.

- nested_crossproduct — Similar to flat_crossproduct but maintains the hierarchical structure of input combinations.

By leveraging these scatter methods, geospatial workflows can efficiently process large datasets in a distributed computing environment. To achieve this on HPC clusters, CWL is executed through Toil WES, which maps the scattered processes onto SLURM job submissions, ensuring optimal utilization of computational resources.

```
(venv3.11) (base) x-gfenoy@login07.anvil:[cybergis] $ TOIL_CHECK_ENV=True
toil-cwl-runner --jobStore=/home/x-gfenoy/TOIL/toil_storage/example/job_store/
$(uuidgen) --batchSystem=slurm         --singularity      --defaultDisk=6G --
maxDisk=8G app-p
ackage.cwl#extrac-inundation-census-tract example-job1.yaml
[2025-02-03T13:12:04-0500] [MainThread] [I] [cwltool] Resolved 'app-package.
cwl#extrac-inundation-census-tract' to 'file:///home/x-gfenoy/cybergis/app-
package.cwl#extrac-inundation-census-tract'
[2025-02-03T13:12:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Importing input
files...
[2025-02-03T13:12:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Importing tool-
associated files...
[2025-02-03T13:12:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Creating root job
[2025-02-03T13:12:05-0500] [MainThread] [I] [toil.cwl.cwltoil] Starting workflow
[2025-02-03T13:12:05-0500] [MainThread] [I] [toil] Running Toil version 7.0.0-d56
9ea5711eb310ffd5703803f7250ebf7c19576 on host login07.anvil.rcac.purdue.edu.
[2025-02-03T13:12:07-0500] [MainThread] [I] [toil.leader] 0 jobs are running, 1
jobs are issued and waiting to run
[2025-02-03T13:12:07-0500] [MainThread] [I] [toil.leader] Issued job 'CWLJob'
extrac-inundation-census-tract.1.process kind-CWLJob/instance-vh_7qnf1 v1
with job batch system ID: 2 and disk: 5.9 Gi, memory: 11.8 Gi, cores: 4,
accelerators: [], preemptible: False
[2025-02-03T13:12:07-0500] [MainThread] [I] [toil.leader] Issued job 'CWLJob'
extrac-inundation-census-tract.0.process kind-CWLJob/instance-nqs8ej8a v1
with job batch system ID: 3 and disk: 5.9 Gi, memory: 11.8 Gi, cores: 4,
accelerators: [], preemptible: False
[2025-02-03T13:13:29-0500] [Thread-4 (statsAndLoggingAggregator)] [I] [toil.
statsAndLogging] Got message from job at time 02-03-2025 13:13:29: CWL step
complete: extrac-inundation-census-tract.0.process
[2025-02-03T13:13:29-0500] [Thread-4 (statsAndLoggingAggregator)] [I] [toil.
statsAndLogging] extrac-inundation-census-tract.0.process.stdout follows:
=========>
        /UndHuj
        Output Directory:  /tmp/Multi_F_Results/N_0
        Temp Directory:  /UndHuj
        Total Dams: 345
        Creating output file that is 363P x 267L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//MH_F_SD01097.tiff
[1/1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//MH_F_SD01097.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//MH_F_SD01097.tiff to destination /UndHuj/MH_F_SD01097_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /UndHuj/MH_F_SD01097.json of format GeoJSON.
        100 - done.
        Creating output file that is 363P x 267L.
```

```
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//TAS_F_SD01097.
tiff [1/1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//TAS_F_SD01097.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//TAS_F_SD01097.tiff to destination /UndHuj/TAS_F_SD01097_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /UndHuj/TAS_F_SD01097.json of format GeoJSON.
        100 - done.
        Creating output file that is 363P x 267L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//NH_F_SD01097.tiff
[1/1] : 0Using internal nodata values (e.g. 0) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//NH_F_SD01097.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//NH_F_SD01097.tiff to destination /UndHuj/NH_F_SD01097_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /UndHuj/NH_F_SD01097.json of format GeoJSON.
        100 - done.
        Archive:  /tmp/tl_2020_46_tabblock20.zip
         extracting: tl_2020_46_tabblock20.cpg
         extracting: tl_2020_46_tabblock20.dbf
         extracting: tl_2020_46_tabblock20.prj
         extracting: tl_2020_46_tabblock20.shp
         extracting: tl_2020_46_tabblock20.shp.ea.iso.xml
         extracting: tl_2020_46_tabblock20.shx
         extracting: tl_2020_46_tabblock20.shp.iso.xml
        SD01097: Step 1, 1/4, Identifying associated regions for multiple
scenarios
        SD01097: Step 1, 2/4, Search states associated
        -- SD01097 impacts 1 States, ['46']
        SD01097: Step 1, 3/4, Extracting GEOID of census blocks
        SD01097: Step 1, 4/4, Assigning geometry to census blocks
        SD01097: Step 2, 1/2, Merging census data to FIM geoid
        SD01097: Step 2, 2/2, Calculating Moran's I and LISA
        Merging results for 1 dams
        Saving results Multi_F_fim.geojson
        Saving results Multi_F_mi.geojson
        Saving results Multi_F_lm.geojson
        Done
<=========
[2025-02-03T13:13:35-0500] [Thread-4 (statsAndLoggingAggregator)] [I] [toil.
statsAndLogging] Got message from job at time 02-03-2025 13:13:35: CWL step
complete: extrac-inundation-census-tract.1.process
[2025-02-03T13:13:35-0500] [Thread-4 (statsAndLoggingAggregator)] [I] [toil.
statsAndLogging] extrac-inundation-census-tract.1.process.stdout follows:
=========>
        /mrUvUX
        Output Directory:  /tmp/Multi_F_Results/N_0
        Temp Directory:  /mrUvUX
        Total Dams: 345
        Creating output file that is 367P x 200L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//MH_F_MS03356.tiff
[1/1] : 0Using internal nodata values (e.g. -9999) for image /vsicurl?list_dir=
```

```
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//MH_F_MS03356.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//MH_F_MS03356.tiff to destination /mrUvUX/MH_F_MS03356_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /mrUvUX/MH_F_MS03356.json of format GeoJSON.
        100 - done.
        Creating output file that is 366P x 199L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//TAS_F_MS03356.tiff
[1/1] : 0Using internal nodata values (e.g. -9999) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//TAS_F_MS03356.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//TAS_F_MS03356.tiff to destination /mrUvUX/TAS_F_MS03356_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /mrUvUX/TAS_F_MS03356.json of format GeoJSON.
        100 - done.
        Creating output file that is 366P x 199L.
        Processing /vsicurl?list_dir=no&empty_dir=yes&url=https://www.hydroshare.
org/resource/1de14e8622564aa5bd645b92d899a201/data/contents//NH_F_MS03356.tiff
[1/1] : 0Using internal nodata values (e.g. -9999) for image /vsicurl?list_dir=
no&empty_dir=yes&url=https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92
d899a201/data/contents//NH_F_MS03356.tiff.
        Copying nodata values from source /vsicurl?list_dir=no&empty_dir=yes&url=
https://www.hydroshare.org/resource/1de14e8622564aa5bd645b92d899a201/data/
contents//NH_F_MS03356.tiff to destination /mrUvUX/NH_F_MS03356_resample.tiff.
        ...10...20...30...40...50...60...70...80...90...100 - done.
        0...10...20...30...40...50...60...70...80...90...Several drivers
matching json extension. Using GeoJSON
        Creating output /mrUvUX/NH_F_MS03356.json of format GeoJSON.
        100 - done.
        Archive:  /tmp/tl_2020_28_tabblock20.zip
         extracting: tl_2020_28_tabblock20.cpg
         extracting: tl_2020_28_tabblock20.dbf
         extracting: tl_2020_28_tabblock20.prj
         extracting: tl_2020_28_tabblock20.shp
         extracting: tl_2020_28_tabblock20.shp.ea.iso.xml
         extracting: tl_2020_28_tabblock20.shx
         extracting: tl_2020_28_tabblock20.shp.iso.xml
        MS03356: Step 1, 1/4, Identifying associated regions for multiple
scenarios
        MS03356: Step 1, 2/4, Search states associated
        -- MS03356 impacts 1 States, ['28']
        MS03356: Step 1, 3/4, Extracting GEOID of census blocks
        MS03356: Step 1, 4/4, Assigning geometry to census blocks
        MS03356: Step 2, 1/2, Merging census data to FIM geoid
        MS03356: Step 2, 2/2, Calculating Moran's I and LISA
        Merging results for 1 dams
        Saving results Multi_F_fim.geojson
        Saving results Multi_F_mi.geojson
        Saving results Multi_F_lm.geojson
        Done
<=========
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.leader] Finished toil run
successfully.
```

```
Workflow Progress 100%|██████████████████████████████| 5/5 (0 failures) [01:38<00:00, 0.05 jobs/
s]
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Collecting
workflow outputs...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-outyfqnlg55/Multi_F_fim.geojson...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-outyfqnlg55/Multi_F_mi.geojson...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-outyfqnlg55/Multi_F_lm.geojson...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-outajnb5eui/Multi_F_fim.geojson...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-outajnb5eui/Multi_F_mi.geojson...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Saving file:///
home/x-gfenoy/cybergis/tmp-outajnb5eui/Multi_F_lm.geojson...
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Stored workflow
outputs
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] Computing output
file checksums...
```

```
{
    "stac": [
        {
            "location": "file:///home/x-gfenoy/cybergis/tmp-outyfqnlg55",
            "basename": "tmp-outyfqnlg55",
            "nameroot": "tmp-outyfqnlg55",
            "nameext": "",
            "class": "Directory",
            "listing": [
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-outyfqnlg55/
Multi_F_fim.geojson",
                    "basename": "Multi_F_fim.geojson",
                    "size": 133506,
                    "checksum": "sha1$1fec430665800a5c637ed36892d391b39c95fd0c",
                    "nameroot": "Multi_F_fim",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-outyfqnlg55/Multi_F_fim.
geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-outyfqnlg55/
Multi_F_mi.geojson",
                    "basename": "Multi_F_mi.geojson",
                    "size": 1488,
                    "checksum": "sha1$11eaf14b4b33a5587e95c18ed3b252d85e862bd1",
                    "nameroot": "Multi_F_mi",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-outyfqnlg55/Multi_F_mi.
geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-outyfqnlg55/
Multi_F_lm.geojson",
                    "basename": "Multi_F_lm.geojson",
                    "size": 423877,
                    "checksum": "sha1$e4ff6b15fe30177882d6fa686f8bfbcbb67cbb6d",
                    "nameroot": "Multi_F_lm",
```

```
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-outyfqnlg55/Multi_F_lm.
geojson"
                }
            ],
            "path": "/home/x-gfenoy/cybergis/tmp-outyfqnlg55"
        },
        {
            "location": "file:///home/x-gfenoy/cybergis/tmp-outajnb5eui",
            "basename": "tmp-outajnb5eui",
            "nameroot": "tmp-outajnb5eui",
            "nameext": "",
            "class": "Directory",
            "listing": [
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-outajnb5eui/
Multi_F_fim.geojson",
                    "basename": "Multi_F_fim.geojson",
                    "size": 20087,
                    "checksum": "sha1$cfb14c04ae8c75eb5c1b0f160bff33fd49f5df77",
                    "nameroot": "Multi_F_fim",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-outajnb5eui/Multi_F_fim.
geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-outajnb5eui/
Multi_F_mi.geojson",
                    "basename": "Multi_F_mi.geojson",
                    "size": 1479,
                    "checksum": "sha1$573512a7e01873e0a5bdfcfeac094431b3c68235",
                    "nameroot": "Multi_F_mi",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-outajnb5eui/Multi_F_mi.
geojson"
                },
                {
                    "class": "File",
                    "location": "file:///home/x-gfenoy/cybergis/tmp-outajnb5eui/
Multi_F_lm.geojson",
                    "basename": "Multi_F_lm.geojson",
                    "size": 195914,
                    "checksum": "sha1$02666ae0d29e8c1a11a5f8dd2fd736baebfabcf9",
                    "nameroot": "Multi_F_lm",
                    "nameext": ".geojson",
                    "path": "/home/x-gfenoy/cybergis/tmp-outajnb5eui/Multi_F_lm.
geojson"
                }
            ],
            "path": "/home/x-gfenoy/cybergis/tmp-outajnb5eui"
        }
    ]
}
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.cwl.cwltoil] CWL run complete!
[2025-02-03T13:13:48-0500] [MainThread] [I] [toil.common] Successfully deleted
the job store: FileJobStore(/home/x-gfenoy/TOIL/toil_storage/example/job_store/
e630922c-ece4-4c92-940a-b21a9369bdc8)
```


Listing C.22 — Run log from toil-cwl-runner with multiple inputs

C.3.5. Performance Evaluation

The performance evaluation of the COG files in HPC environments was conducted using a set of benchmarks and metrics to assess the efficiency and scalability of COG in geospatial data processing workflows. The evaluation included the following key aspects:

- Speed: Measure the data read/write speeds for accessing and processing COG files.
- Scalability: Assess how well COG files handle increasing data volume and complexity.
- Efficiency: Evaluate computational efficiency in terms of resource usage (CPU, memory).

The evaluation was conducted using open-source geospatial libraries and software to work with COG files. Benchmarking tools were used to collect CPU and memory resource utilization metrics and timing metrics for performance evaluation. The HPC environment provided by I-GUIDE was utilized for this investigation.

The performance evaluation included a comparison of COG's performance against traditional GeoTIFF files in terms of speed, scalability, efficiency, and functionality. The evaluation results provide insights into the benefits and drawbacks of using COG files in HPC environments and recommendations for adopting COG in geospatial data processing workflows.

C.3.6. Conclusion

If using Scatter permits parallelization of tasks, the workflow was encapsulated as a single entity when splitting a workflow into smaller parts as some tasks that run internally to the workflow could have been executed in parallel. For instance, executing every scenario in parallel rather than running the whole workflow for each individual dam.

The workflow that was chosen for this activity should have been split into smaller parts for enhancing scalability and optimizing performance. Also, the workflow used different formats that are accessed at every run online. Further, some datasets are provided in zipped shapefiles format. Having the data available in a more standard format such as that used in the OGC API — Features service with a backend based on GeoParquet would have been of interest.

With the system in place, toil to execute the CWL workflow, and the capability to ask for statistical information about CPU time, memory consumption, disk usage, and so on; with the `toil —stats ` command, it is possible to compare the processing time of the workflow with inputs from the initial GeoTIFF format to COG with the different tilesize and compression options.

The next step should be to evaluate workflows with more granularity to ensure that the task can be clearly identified within a workflow execution. This will allow for better monitoring and optimization of the workflow execution.

D

ANNEX D (NORMATIVE) HPC OPTIMIZED GEOZARR COMPONENT

ANNEX D
(NORMATIVE)
HPC OPTIMIZED GEOZARR COMPONENT

D.1. Introduction

The GeoZarr format enables content creators to store both vector and raster data within a single structure. Although you can store vector data in a GeoZarr container, GeoZarr is typically used for Raster datasets. Zarr/GeoZarr was designed to store multiple dimensions that correspond to the same pixel or object. Unlike other formats that limit storage to just one type of data, GeoZarr can handle multiple types simultaneously, making it highly versatile.

D.1.1. Example Use Case:

Example: One use case involves analyzing a series of images to detect changes over time. GeoZarr files could store features correlated through the time dimension to the series of images. However, the time dimension is typically not included in a GeoZarr file. Using a Geographical Information System (GIS), both the imagery and the features can be synchronized based on the time dimension, highlighting changes over time within a unified dataset.

D.1.2. Advantages of GeoZarr:

1. **Data Organization:** GeoZarr files support the creation of groups. These groups can contain subgroups and/or arrays.
2. **Scalability:** Arrays can be divided into chunks. This chunking allows content producers to manage Zarr files in units that optimize both processing and scalability as new data becomes available.

D.2. Methodology

This section outlines the methodology for evaluating the capabilities of using GeoZarr in High Performance Geospatial Computing (HPGC) environments. The evaluation process included data

collection, selection criteria, evaluation criteria, tools and techniques, and analysis methods. The evaluation examined a use case for extracting Area of Interest (AoI) selections from imagery data and adding the selections as a result set to the GeoZarr dataset container.

D.2.1. Data Collection

1. **Data Sources:** Hydroshare NetCDF data from https://www.hydroshare.org. The Hydroshare Dataset is licensed under a Creative Commons Attribution CC BY 4.0 License.
2. **Data Types:** The dataset includes a set of multidimensional time series data.
3. **Data Volume:** The exploration looked at one NetCDF file.

D.2.2. Selection Criteria

1. **Relevance:** For benchmarks, the imagery dataset should reference a similar area with collections from various times.

D.2.3. Evaluation Criteria

D.2.3.1. Performance Metrics:

Scalability: Assess how well GeoZarr handles increasing data AOI complexity.

D.2.3.2. Functionality:

- **Data Access:** Test the ease of data access and retrieval.
- **Data Management:** Evaluate capabilities for data updating, indexing, and partitioning.

D.2.4. Tools and Techniques

1. **Geospatial Tools:** The benchmark participants supplied open-source libraries and software to work with the data.
2. **HPC Environment:** I-GUIDE provides the High-Performance Computing environment for the benchmark.

D.2.5. Analysis Methods

1. **Qualitative Analysis:**
    - Document observations regarding functionality and ease of use.
    - Examine the benefits and cons for the use of processing on an HPC.
2. **Comparison Analysis:**
    - Identify strengths and weaknesses relative to the evaluation criteria.

D.3. Experimentation

To evaluate the suitability of using the HPC for processing, the benchmark applied the following steps:

1. Stage the collected imagery on a single machine or node.
2. Install the software the contributor provides as a container on the machine or node in a container registry.
3. Run the software which performs the following tasks:
    - Convert the NetCDF data into a Zarr dataset.
        - Iterate over each image in the time series and add the image to the Zarr dataset.
            - Use multiprocessing to process various rows of the image on a single or series of compute nodes.
        - Modify the time index to expand the time dimension with the addition of each image in the dataset.
            - Maintain the time dimension in ascending order from earliest to most recent times.

D.4. Evaluation

The tests used in the OGC Testbed-20 HPC benchmark activity resides in a GitHub repository [geozarr-lib]. The repository contains an API for writing GeoZarr datasets and is licensed using an Apache 2 Open Source license. The code can be found at https://github.com/leedahl/geozarr. The benchmark used the Python `unittest` [25] module to run an integration test stored in the repository at tests/int/test_create_dataset.py. When examining the unit test, the code defines a test dataset to create, the schema for the dataset, and a NetCDF file to convert to a GeoZarr dataset.

```
        resource_path = f"{'/'.join(path.split(__file__)[0].split('/')[0:-1])}/
resources"
        test_netcdf_path = f'{resource_path}/SWE_time.nc'

        base_path = f'file://{path.split(__file__)[0]}/test_results'
        file_path = f'{base_path}/create_dataset.zarr'
        with GeoZarr(file_path, 'x') as geo_zarr:
            # Load sample data
            with xr.load_dataset(test_netcdf_path) as dataset:
                # Define schema
                chunks = [1]
                chunks.extend(list(dataset['SWE'].shape[1:]))
                schema = {
                    'grid': {
                        'upperLeft': [433570.9001397601, 4663716.608805167],
                        'unitSize': 800,
                        'crs': dataset['transverse_mercator'].attrs['spatial_
ref']
                    },
                    'global_attributes': {'conventions': 'CF-1.11'},
                    'SWE': {
                        'name': 'SWE',
                        'attributes': {
                            'long_name': 'Snow water equivalent',
                            'units': 'm'
                        },
                        'shape': dataset['SWE'].shape,
                        'chunks': chunks,
                        'dtype': np.float64,
                        'dimensions': [
                            ('time', 'time',  'ns', np.dtype('<M8[ns]'),
DimensionType.DIMENSION_VALUE),
                            ('y', 'projection_y_coordinate', 'm', np.float64,
DimensionType.COORDINATE_Y),
                            ('x', 'projection_x_coordinate', 'm', np.float64,
DimensionType.COORDINATE_X)
                        ]
                    }
                }

                geo_zarr.schema = schema
```

Listing D.1

Examining the above code, note that the schema contains three parts: grid, attributes, and data properties. The step after defining the schema involves inserting data into the new dataset. When inserting data, the code first associates a schema index with an empty set of values to

the defined dataset defined in the schema properties (For example the time index with /SWE/time). Next the code cycles through a series of dataset points or objects from the NetCDF file and inserts them into the dataset (see the code below).

```
                # Insert Data
                geo_zarr.set_index('/SWE/time', dataset.get_index('time').values)

                loop = asyncio.get_event_loop()
                futures = set()
                for index in range(dataset.variables['SWE'].sizes['time']):
                    data: np.ndarray = dataset.variables['SWE'].values[index]
                    dim_index = [('/SWE/time', dataset.get_
index('time').values[index])]
                    futures.add(geo_zarr.insert('/SWE/SWE', data, dim_index))

                loop.run_until_complete(asyncio.gather(*futures))
                loop.close()
```

Listing D.2

Notice that an async event loop manager is created. The GeoZarr API defines a Python Async Method to insert data into the dataset in parallel. At the moment, this code is multithreaded and does not support distributed calls between nodes of an HPC. The code does run on an HPC. However, the code will only use the processors on the node that the code is run on. The `loop.run_until_complete` command initiates an event loop and executes all the inserts in parallel.

D.5. Results and Discussions

D.5.1. Summary

Running the benchmark took around three seconds and created 2,184 data files. The dataset contained three indexes (dimensions) for time, x coordinate and y coordinate with a value of temperature. The data was partitioned by the time dimension as defined by the chunk property in the dataset schema. All the data files were 204 bytes in size. A coder could choose different partitioning schemes to create fewer or more data files.

A drawback of using Zarr formatted datasets is that the number of files in a large dataset could become very large. A large number of files can affect reading of the dataset by reducing I/O performance. In addition, a large number of small files leads to inefficient partitioning on storage devices. However, the benefit of using Zarr formatted datasets is that small partitions support adding or updating data easily without massive amounts of memory, compute time, or I/O. Another advantage to reading Zarr formatted datasets is that an application reading from the dataset can choose to read only a few partitions. In this example, an application may only be interested in observing temperatures from a small subset of time periods. Thus, the application only needs to read a few files from the underlining storage device.

During the examination of Cloud Optimized File Formats, the OGC Testbed-20 participants evaluated two APIs for working with Zarr files. Python XArray is easy to use. However, XArray seems to require knowing how large the dataset will be and requires that a coder load all

the data at once. This is okay for smaller datasets and datasets that do not change overtime. However, XArray does not seem to be a good choice for larger datasets. The OGC Testbed-20 participants considered the Python Zarr module. The code in the Zarr module is more flexible. Nevertheless, participants agreed that this module is more difficult than it needed to be. Therefore, the participants created an open-source API for writing GeoZarr files as described in the evaluation section above.

D.5.2. Recommendations

The OGC Testbed-20 participants provide the following guidance and recommendations for using the `geozarr` API [geozarr-lib] on HPC environments:

- Coders should be familiar with the data and the application use case being developed as data partitioning is important for application performance.
    - Larger datasets that will grow over time require the ability to add data. Therefore, keeping partition sizes to a level that fits into memory of processing nodes is important.
    - Smaller partitions result in fewer changes when inserting data.
    - Understand the use case because partitions can be created across all dimensions of the dataset.
        - When reading raster data from a time-series, it may be beneficial to tile the raster data in addition to partitioning by date.
        - If the application requires reading subsets of a large raster file, tiling its data array with GeoZarr makes it quicker to read specific parts of the raster.
- Coders should also understand the following when distributing the creation and reading of a dataset is advantageous.
    - With large datasets can use partitioning to distribute the processing of creating and reading the dataset.
    - Writing data from a stream such as data from a collection device as part of the Internet of Things (IOT) can benefit from feeding the writing of data to the Zarr dataset in a distributed manner.
        - Such an application requires keeping the partition sizes small so preprocess the data to queue partition chunks of data for a writing process.
        - If the adding of streaming data requires updating dimension values such as adding dates, this task needs to use some sort of locking method to serially update the dimension values.

In general, the participants felt that GeoZarr is well-suited for processing on High Performance Computing (HPC) clusters. Distributed programming and parallel processing will support the

efficient writing and reading of large or streaming datasets. The main issue that coders will face is knowing how to partition the data. Large partitions support better compression rates. However, smaller partitions lend themselves to distributing the processing to HPC cluster nodes.

D.6. Future Investigations

The development of Zarr version 3 introduces several intriguing new features that future investigations could exploit. While Zarr version 3 is relatively new and currently lacks widespread implementation, future investigations may see the development of tools that take advantage of these new features.

In addition, the OGC Testbed-20 participants ran out of time to completely test distributed writing operations on the HPC. The participants also ran out of time for creating code to read from GeoZarr files. So future work could focus on greater distributed testing and also coding a "read" Application Programming Interface (API). Additionally, the Geo Extensions to Zarr were not evaluated in OGC Testbed-20. Most of the Geo Extension center on portrayal. Visualization was not the focus of the Testbed-20 HPC task. Therefore, future work could look at how to create and deliver the portrayal aspects of GeoZarr files.

D.6.1. Sharding Index

One such feature is the sharding index [27]. Sharding enables storing very large, chunked arrays, which currently would have inefficient access characteristics or might even be impossible to store as generic Zarr arrays. The significance of sharding indexes in Zarr stems from the limitations imposed by storage devices and operating systems on the number and size of files stored. In particular, GeoZarr often deals with small chunk sizes, leading to the creation of numerous small files. While this is advantageous for the distribution and modification of datasets in High Performance Computing (HPC) environments, it poses challenges related to storage and operating system constraints on cluster nodes.

The Zarr version 3 sharding index mitigates the issue of small files by allowing a single file to hold multiple chunks of data. This approach is similar to how GeoTIFF files support internal tiling. By sharding chunks, data creators can organize data to achieve the best of both worlds: Small chunks and appropriately sized files. The term "appropriately sized" indicates that data creators can group chunks into files optimized for storage devices without combining all chunks into a single file.

This method allows for better distribution of processing while also reducing the complexity involved in adding and modifying data.

E

ANNEX E (NORMATIVE) HPC OPTIMIZED GEOPARQUET COMPONENT

ANNEX E (NORMATIVE) HPC OPTIMIZED GEOPARQUET COMPONENT

E.1. Introduction

The extension to the Parquet format, GeoParquet, enables creators to store vector information as well as attributes about the vector data. Unlike other storage formats, GeoParquet stores data in a columnar format rather than a row format. The format partitions data by both columns and row groups. In addition, further partitioning is available by creating a page size within the row group. With the GeoParquet format, a data creator can index the data by specifying bounding boxes for row groups and pages. With other attributes, a data creator can index the attribute date by specifying a maximum and minimum value.

E.1.1. Example Use Case:

Example: One use case for using a GeoParquet file is to catalog tiled GeoTIFF tile datasets into bounding boxes for each tile and each tiled dataset image. The GeoParquet dataset contains metadata such as a row identifier, the tile size, ground sample distance, the polygon for the bounding box, whether the polygon represents the image or a tile in the image datasets and a filename for the tile or image. Such a catalog would be used for image inventory queries to locate tiles and images that are of interest.

E.1.2. Advantages of GeoParquet

The advantage of using GeoParquet on a High-Performance Computing (HPC) cluster is the ability to partition data. The more the data can be partitioned, the better performance a distributed database controller can achieve. The fact that data is partitioned by columns allows the data curator to add additional columns without having to reindex any existing files. Column portioning in GeoParquet allows a data curator to put column/row group data in separate files. This chunking of data into smaller files is suited to the processing of data on distributed nodes of the HPC.

The indexing concept in GeoParquet enables a database engine to skip the reading of data files that are not relevant to a query. In addition, GeoParquet does not store null data making the storage data more compact. When a database controller is asked to store null data, the

controller stores a Boolean bit in a definition level indicating if the value exists or is null. Thus, the controller only needs to adjust the definition level when adding null values.

E.2. Methodology

This section outlines the methodology for evaluating the capabilities of GeoParquet within High Performance Geospatial Computing (HPGC) environments. The evaluation process included data collection, selection criteria, evaluation criteria, tools and techniques, and analysis methods. The evaluation examined a use case for extracting bounding boxes from the imagery data and adding them as a result set into the GeoParquet dataset.

E.2.1. Data Collection

1. **Data Sources:** SpaceNet.ai data for challenge number 8 curated by Maxar Technologies for an area in Dernau Germany. The dataset is licensed on Creative Commons Attribution ShareAlike 4.0 International License.
2. **Data Types:** The dataset contains a sparse array of a tiled pre-flood image of Dernau Germany.
3. **Data Volume:** The dataset contains 202 tiles over the area.

E.2.2. Selection Criteria

1. **Relevance:** The data contains a sparse array of tiled data that can be read and cataloged with similar sized tiles.

E.2.3. Evaluation Criteria

E.2.3.1. Performance Metrics:

Scalability: Assess how well GeoParquet handles increasing data sizes.

E.2.3.2. Functionality:

- **Data Access:** Test the ease of data access and retrieval.
- **Data Management:** Evaluate capabilities for data updating and partitioning.

E.2.4. Tools and Techniques

1. **Geospatial Tools:** The benchmark participants supplied open-source libraries and software to work with the data.
2. **HPC Environment:** I-GUIDE provided the High-Performance Computing environment for this investigation.

E.2.5. Analysis Methods

1. **Qualitative Analysis:**
   - Document observations regarding functionality and ease of use.
   - Examine the advantages and disadvantages of doing the GeoParquet processing on an HPC platform.
2. **Comparison Analysis:**
   - Identify strengths and weaknesses relative to the evaluation criteria.

E.3. Experimentation

To evaluate the suitability of using the HPC for processing, this benchmark performed the following steps:

- Experiment 1
  1. Stage the collected imagery on the HPC.
  2. Install the software the participants provided as a container on the machine or node in a container registry.
     - Tiff Mapper Container: An application that reads bounding boxes from an imagery tile set and adds them to a GeoParquet dataset.
     - GeoParquet Container: A library that contains the GeoParquet API code.
  3. Start up Ray [28] nodes on HPC nodes using the container.
  4. Run the Tiff Mapper software which performs the following task:
     - Start a node to run the HPC tile mapper.

- Connect to the Ray cluster. A Ray cluster is a group of worker nodes connected to a head node that work together to perform computations in parallel.
- Iterate over the list of GeoTIFF files within tile set.
    - Extract the bounding box of each GeoTIFF file.
    - Calculate the min / max coordinates of the imagery dataset with each new bounding box added to the dataset.
    - Sum up the ground sample distance values for each GeoTIFF tile image.
    - Insert the record for the current GeoTIFF file.
- Create the bounding box for the min / max coordinates collected during the iteration of GeoTIFF tiles.
- Create the average ground sample distance value for the GeoTIFF imagery dataset.
- Insert the record for the GeoTIFF imagery dataset collection.
- Iterate over 256 by 256 tiles over the bounding box of the GeoTIFF imagery dataset collection.
    - For each tile, calculate the bounding box of the 256 x 256 pixel tile using the average ground sample distance as the pixel size.
    - For each tile, insert a record in the GeoParquet dataset to represent a tile in a tiled dataset collection.

- Experiment 2

    1. Stage the collected imagery on the HPC.
    2. Install the software the benchmark participants provided as a container on the machine or node in a container registry.
        - Tiff Mapper Container: An application that reads bounding boxes from an imagery tile set and adds them to a GeoParquet dataset.
    3. Run the Tiff Mapper software which performs the following task:
        - Start a HPC node to run the tile mapper on.
        - Create a ray cluster that runs on the current node off the HPC.
        - Iterate over the list of GeoTIFF files within a tile set.
            - Extract the bounding box of each GeoTIFF file.

        - Calculate the min / max coordinates of the imagery dataset with each new bounding box added to the dataset.
        - Sum up the ground sample distance values for each GeoTIFF tile image.
        - Insert the record for the current GeoTIFF file.
    - Create the bounding box for the min / max coordinates collected during the iteration of GeoTIFF tiles.
    - Create the average ground sample distance value for the GeoTIFF imagery dataset.
    - Insert the record for the GeoTIFF imagery dataset collection.
    - Iterate over 256 by 256 tiles over the bounding box of the GeoTIFF imagery dataset collection.
        - For each tile, calculate the bounding box of the 256 x 256 pixel tile using the average ground sample distance as the pixel size.
        - For each tile, insert a record in the GeoParquet dataset to represent a tile in a tiled dataset collection.

E.4. Evaluation

The Testbed 20 participants ran two experiments for the HPGC thread. The purpose of the first experiment was to assess running the tile mapper against a distributed cluster of nodes on the HPC. The purpose of the second experiment was to assess multiprocessing on a single node of the HPC. The code used for the experiments can be found in the following GitHub repositories:

- TIFF Mapper: https://github.com/leedahl/tiff_mapper
- GeoParquet API: https://github.com/leedahl/geoparqet_library

The TIFF Mapper and GeoParquet API use Ray.io distributed framework to perform multiprocessing and/or distributed processing. The participants were familiar with the Ray.io distributed framework. To shorten development time, the participants choose to implement the code using the Ray.io framework. There are many ways to create a set of distributed compute cluster nodes. However, the support for creating distributed compute cluster nodes on the HPC was a bit limited. To ease starting Ray on the HPC, the participants created a throwaway script to start the nodes. To limit side effects, the participants started the shell with a 5-minute execution limit for both worker nodes and the master node on the HPC.

E.5. Results and Discussions

The first experiment failed to produce the desired results. The Ray.io Framework was able to connect to worker nodes on the HPC. However, trying to pass code and parameters between nodes of the cluster seemed to lock up the nodes. The 5-minute execution time limited the negative effect of using 100% resources on the nodes and lockup access to the nodes. Determining the exact cause of the lockup was not achieved. However, the participants speculated that running the Ray.io framework in a container versus running it on the bare metal machine could be the cause. More work would be required to determine how to use the Ray.io distributed framework on the HPC within a container application.

The second experiment was successful. Creating a Ray cluster of processes on a single node using multiprocessing within a single container worked. This is one reason the participants concluded that communication between nodes to access different container instances caused the failure in the first experiment. If multiprocessing within a container using Ray.io framework works fine but does not work between containers, then it stands to reason that something with the containerization between nodes prevented the proper communications.

Creating the GeoParquet dataset was quick, and the distributed insert process worked well.

More work is required to investigate other distribution frameworks and containerization to determine how to scale on an HPC. However, the fact that the test did create multiple files for each column and was able to use multiprocessing to insert columns and records at the same time shows the potential for HPC applications.

E.5.1. Summary

E.5.2. Recommendations

The benefit of containerization is that coders can keep the containers up to date with new toolsets and everyone can have customized toolsets that meet their needs. The drawback of containerization is that networked applications are harder to write and work with. The initial results of the Testbed-20 investigations showed that while a status display of the ray cluster shows multiple nodes attached, the nodes were not able to communicate correctly. One recommendation would be to run the application on the HPC hardware without containerization. However, with the hardware/software provided for this benchmark, the code needed to be refactored to run on an older version of Python. The other option is to update the software stack on the HPC. However, that could affect other current projects and probably would not be considered. So, this leaves the coders with the option of either figuring out what the communication barrier is or looking at other forms of distributed programming. Then, the only real option that presents itself with the current code is to run the container on a single node. This works and still takes advantage of multiprocessing.

Partitioning and compression are other issues to consider for data curators. Compressing small amounts of data does not provide much storage savings. In addition, the storage hardware has its own partitioning scheme. One recommendation for the curator is to define page sizes that

match the storage partitioning scheme and are big enough to offer real compression value. The curator should also consider the size of the row group in records. Knowledge of the potential storage size of a record along with the page size should allow a curator to determine how many records to put into a row group. The curator needs to weigh the partition size against the need for parallel reading/writing and the amount of change that will occur with the dataset. Row Groups for a column can be stored in separate files. Smaller partitions tend to allow for better distribution access in distributed programming. In addition, smaller partitions mean less refactoring of data when records are inserted, updated or deleted. However, if the data is fairly static and queries return large record sets, the curator will then want to create large partitions that can be compressed.

Another consideration for fairly static datasets is record grouping. Since GeoParquet deals with spatial data, the curator may want to define row groups that group spatially near data together. This is because indexing of spatial data is done with a bounding box. A large bounding box with different row groups overlapping each make for very inefficient queries. Considering “spatial” awareness to the data also applies to the paging scheme within the row group as each page can have its own bounding box index. The one exception to this is when you have a large volume of data where all features are spatially near each other. In this situation a data curator may want to have some overlapping bounding boxes as multiprocessing or distributed processing could query several row groups and pages at the same time.

The OGC Testbed-20 HPGC task participants recommend the use of HPC clusters for processing GeoParquet data. The distributed and multiprocessing characteristics of an HPC make HPC clusters a good candidate for processing large GeoParquet datasets. The issue comes in the curation of data to determine record placement and partition sizes.

E.6. Future Investigations

The GeoParquet format is an evolving standard. As such, new features are constantly being added. In fact, one such feature is the ability to create an index for map of data in a single column. This is a relatively new concept in GeoParquet which does not yet have wide adoption. Future work could explore this concept. In a similar vein the GeoParquet format supports the storage of complex, repeating data structures such as maps and lists. This aspect was not explored in OGC Testbed 20. Future work should look at complex data structures and how they are indexed and partitioned.

The issues experienced with using Ray.io distributed framework require further examination. Is there a way to make Ray.io work on the HPC implementation within container applications? Are there other distributed frameworks that would work better on an HPC? Future work could explore other frameworks or dig deeper into Ray.io to determine a way to make it work. Perhaps trying to directly add Slurm support within the container application could be a future work item.

In general, there are more features of GeoParquet which have not yet been added to the open source libraries created for this Testbed experiment. To continue working on improving the open source code by adding these features along with some of the new concepts would create a more useful tool for HPGC work. Additional improvements to the distribution of the work are also

needed to make these tools more valuable for HPC work. The main focus of future work should look at adding more support and fixing the issues with distribution of processing.