***In* Tools and Techniques for High Performance Computing. HUST 2019, SE-HER 2019, WIHPC 2019. Communications in Computer and Information Science, vol 1190, pp 190-204**
2019
Eds Juckeland G., Chandrasekaran S., Springer
ISBN 9783030447274
https://doi.org/10.1007/978-3-030-44728-1_12
https://archimer.ifremer.fr/doc/00597/70946/

The Pangeo Ecosystem: Interactive Computing Tools for the Geosciences: Benchmarking on HPC

Odaka Tina [1], Banihirwe Anderson [2], Eynard-Bontemps Guillaume [3], Ponte Aurelien [1], Maze Guillaume [1], Paul Kevin [2], Baker Jared [2], Abernathey Ryan [4]

[1] Laboratory for Ocean Physics and Satellite Remote Sensing UMR LOPS, Ifremer, Univ. Brest, CNRS, IRD, IUEM Brest, France
[2] National Center for Atmospheric Research, Boulder, CO, USA
[3] CNES Computing Center team Centre National d'Etudes Spatiales Toulouse, France
[4] Lamont Doherty Earth Observatory, Columbia University, New York, USA

Abstract :

The Pangeo ecosystem is an interactive computing software stack for HPC and public cloud infrastructures. In this paper, we show benchmarking results of the Pangeo platform on two di erent HPC sys- tems. Four di erent geoscience operations were considered in this bench- marking study with varying chunk sizes and chunking schemes. Both strong and weak scaling analyses were performed. Chunk sizes between 64MB to 512MB were considered, with the best scalability obtained for 512MB. Compared to certain manual chunking schemes, the auto chunk- ing scheme scaled well.

Keywords : Pangeo, interactive computing, HPC, cloud, benchmarking, Dask, Xarray

1 Introduction

In the geosciences, simulation of physical systems has long been the focus of high-performance computing. Thanks to the excellent scaling properties of geoscientific simulations, scientists can now easily output petabytes of data, which together with the massive increase in the volume of observational data, is leading to a crisis for traditional data analytics workflows. In this community, traditional methods of analysis depend upon serial, non-scalable tools, such as the NetCDF Operators (NCO)[1], the NCAR Command Language (NCL)[2], or serial MATLAB and Python scripts. Alternatively, each scientist had to develop parallel (*e.g.*, MPI) applications to perform specialized analysis on particular datasets, a task that is time-consuming, error prone, and leads to duplication of effort

across the community. These methods of analysis are so time-consuming that scientists have accepted, for many years now, a *batch processing* style for conducting analysis, where the scientist spends considerable effort and time to write a data analysis script that is then submitted to a traditional HPC batch queuing system, such as PBS Pro [3] or SLURM[4]. These batch jobs can take hours to days to complete. This kind of batch-style data analysis (with non-scalable tools) interrupts the natural, iterative nature of the scientific process of data exploration. The geoscience community needs scalable tools that can free scientists to explore their data interactively, and it is to this end that the Pangeo [5–7] community exists.

2 Pangeo & the Pangeo Platform

Pangeo is a community devoted to the development of an ecosystem of interoperable, scalable, open source tools for interactive data analysis [8]. The community is diverse, comprising of members from traditional HPC as well as cloud computing backgrounds, scientists and technologists, and involving both industry and academia.

The Pangeo framework has allowed several scientific results already. Yu et al [9], for example, processed global high resolution numerical simulation outputs of the ocean circulation (a 30 TB dataset) in order to quantify its frequency content and compare it with actual observations. The analysis required non-trivial rechunking of a large dataset which was achieved on an HPC platform with a remarkably light amount of code [10].

The Pangeo platform consists of five components: (a) a thin user interface, such as JupyterLab or Jupyter Notebook [11], (b) a data model, such as Xarray [12] or Iris [13], (c) a scalable computing system, such as Dask [14, 15] or Spark [16], (d) a scalable storage system, such as a parallel filesystem or object storage, and (e) a resource management system, such as an HPC batch job scheduling system or Kubernetes. Exactly which choice you make for each component of the platform depends on how you can access the underlying computing system, the data you wish to analyze, and whether you are running the platform on a traditional HPC system or in the public cloud.

On HPC systems, Pangeo is used mainly with Dask. Dask's computing system is based on a central Dask scheduler and multiple Dask workers. The Dask scheduler orchestrates parallelisation tasks performed by Dask workers. Each parallel task assigned to the Dask workers is based on a 'chunk' of grided data in Xarray datasets. Users specify the size and shape of the 'chunk' and how many (and what kind of) Dask workers to provide to Dask. Then, Dask takes care of the parallelisation automatically. Dask-Jobqueue [15, 17] works with traditional HPC job scheduling systems to launch Dask workers interactively, providing both fixed and adaptive scaling capabilities.

For the purposes of this paper, we consider two specific deployments of the Pangeo platform on HPC systems: (i) the HAL system at CNES [18], and (ii) the Cheyenne system at NCAR's Wyoming Supercomputing Center (NWSC) [19].

The benchmarks performed for this paper consider only interactive compute (*i.e.*, no I/O), and so we only concern ourselves with the data model and scalable compute components of the Pangeo platform.

3 HPC Deployments of the Pangeo Platform

As mentioned in the previous section, and for the purposes of this paper, an HPC Pangeo deployment is distinguished from a cloud-based deployment by the use of a Pangeo Python environment (containing Xarray, Dask, and Dask-Jobqueue) and an HPC batch job scheduling system. Some HPC centers deploy the JupyterHub [20] service, which provides a platform for authenticating users and launching Pangeo Jupyter Notebooks on the remote HPC system.

3.1 Hal

CNES Cluster (Hal) Architecture Hal is an intermediate size HPC cluster, with about 460 nodes, 12,000 cores and a 8.5 PB Spectrum Scale Storage. Nodes and storage are interconnected with Infiniband at 56 Gb/s, and the storage system provides a bandwidth up to 100 GB/s.

Benchmarks for this paper were run on Lenovo compute nodes installed in Hal in 2017 with Intel Broadwell CPUs (2x E5-2650 per node, 24 cores per node) and 128 GB of RAM. Hal has several powerful frontal nodes equipped with more RAM and more powerful CPUs. Standard HPC users use the compute nodes by logging on to the frontal nodes with ssh, develop their applications, then submit their jobs on the command line through PBS Pro. Hal also provides Virtual Machines (VMs) configured as cluster clients. These VMs are integrated into the HPC's network, enrolled in its LDAP directory, mounting the GPFS file system through NFS, and have a PBS client installed and configured. Specific projects or groups of users can ask for one of these VMs in order to have their own environment upon cluster access.

Pangeo Deployment On Hal, JupyterHub was deployed on a VM cluster client within a Conda environment. In order to launch the JupyterHub service, a systemctl service file was set up. ProfileSpawner [21] and BatchSpawner [22] are used to provide a selection of resource profiles for users through a web interface (*e.g.*, number of CPUs, amount of memory), and to start the user's Jupyter Notebook server in a batch job on HPC nodes. The Conda environment providing Pangeo's Python ecosystem Conda was copied from Pangeo's Docker images[5], installed and configured as a Jupyter kernel.

Lessons Learned On the admin side, JupyterHub is the most complex component of the Pangeo deployment, but it was still relatively easy to set up. There is a lack of complete integration, like a provided service file for main linux distributions. BatchSpawner was not compatible with the latest versions of PBS

Pro, which resulted in some Pull Requests to the BatchSpawner codebase. The job script used by BatchSpawner was modified so that users could easily add custom shared kernel folders and configure the Python environment from which the Jupyter Notebook server is launched. Since the installation of JupyterHub in 2018 October, *i.e.* one year ago, more than 100 accounts out of 800 active accounts on Hal have used the service at least once, and nearly 50 accounts use on a weekly basis. About a quarter of JupyterHub users are using Dask for workload distribution. The principal feedback we've obtained on Dask is that it's really easy to start using it, but that it can be challenging to debug or optimize when problems scale up. Distributed computing may look simple, but understanding it and doing it well will always need some expertise, hence this benchmark to determine optimal parameters for common operations.

3.2 Cheyenne

NCAR Cluster (Cheyenne) Architecture Cheyenne is a 5.34 petaflops peak, high-performance computer. It features 145,152 Intel Xeon "Broadwell" processor cores in 4,032 dual-socket nodes (36 cores per node) and 313 TB total system memory (64 GB/node on 3,168 nodes and 128 GB/node on 864 nodes). Cheyenne uses Mellanox EDR InfiniBand in a partial 9D Enhanced Hypercube single-plane interconnect topology with a 25 GB/s bidirectional per link bandwidth. Standard users access the Cheyenne system via ssh with LDAP authentication to 6 dual-socket "Broadwell" login nodes with 256 GB memory/node. Resource management on Cheyenne is provided through PBS Pro.

A separate cluster, named Casper, exists for data analysis and visualization. The Casper system, procured from PCPC Direct, Ltd., is comprised of 28 Supermicro nodes featuring Intel Skylake processors (36 cores per node). Twenty (20) Casper nodes provide 384 GB of RAM for general purpose data analysis and visualization. Six (6) Casper nodes are high-memory nodes with 768 GB of RAM, and two (2) Casper nodes are login nodes. NCAR's JupyterHub provides access to both the Cheyenne and Casper systems, though the benchmarks for this paper where run only on Cheyenne.

Pangeo Deployment Users can access Cheyenne's Pangeo deployment through an experimental JupyterHub deployment running on one of the Cheyenne login nodes, in a setup similar to CNES's Hal JupyterHub. Users are also allowed to launch their own personal installations of JupyterLab over ssh tunnels. NCAR is using this experimental JupyterHub deployment to assess how best to deploy an officially supported JupyterHub for the follow-on machine to Cheyenne in 2021.

Lessons Learned Over the last year, we have made several observations that will help with agility, stability, and upgrades of the JupyterHub service in the future. We have learned that it is extremely beneficial to provide a single access point for the user community with a single web address. Leveraging a reverse proxy has really helped with this but not without difficulties. One issue was

being too restrictive when proxying WebSocket connections as Jupyter applications can heavily rely on the protocol upgrade to function properly. Secondly, as data grows, the size of the Jupyter Notebooks increases as well, necessitating special attention to configuration and sizing of buffering capabilities on the reverse proxy. Additionally, Jupyter, and projects around Jupyter, move quite quickly, and therefore upgrades are expected to be delivered at a more rapid pace than other systems-based software. Currently all JupyterHub installations are kept around to revive them if needed. Separation of services is also critical. The reverse proxy, the different JupyterHub instances, and the site-provided kernels all run in different environments to allow each component to be updated individually. The site-provided kernels remain in a fixed state after they are validated to encourage as much repeatability as possible. Finally, the JupyterHub instances all run within a containerization environment called Inception that allow us to run with necessary changed system configuration files on already existing hardware as part of the machines. The site service has been well adopted and provided great value to workshops and hackathons that have taken place.

In the future, there are plans to increase database resilience by moving to PostgreSQL, or another potentially compatible database, and implementing better telemetry and utilization metric tracking. Finally, we are planning additional investigations into adaptively balancing the use of traditional batch schedulers (*e.g.*, PBS Pro, SLURM) for both batch jobs and interactive computing (via the JupyterHub and Dask-Jobqueue).

4 Benchmark of Pangeo on HPC

4.1 Benchmark Method

During this study, we varied our benchmarking computations in following ways:

- Dask chunk size, S_{chunk},
- cluster size (number of HPC nodes), N_{node}, and
- the chunking scheme used for Dask arrays.

To be able to compare the performance between different architectures we placed only one Dask worker with one thread on each HPC node. On Hal (Sec. 3.1), Dask-Jobqueue was used to submit jobs to PBS Pro job scheduler reserving 24 cores (the entire node) and 128 GB of memory for each Dask worker, ensuring that no other jobs would run on the node for benchmark. On Cheyenne (Sec. 3.2), Dask-Jobqueue was used to submit jobs to the `regular` queue, which reserves entire nodes for each job.

Dataset For each chunk size, we created a random `float64` Xarray dataset called `ds` with the following 3 coordinates: `time`, `lon` (longitude) and `lat` (latitude). This synthetic dataset mirrors the structure of many real datasets in

weather and climate research, such as satellite products or climate model outputs. The size of the total dataset S_{total} is a function of chunk size S_{chunk}, cluster size N_{node} and number of chunks per node F according to:

$$S_{total} = S_{chunk} \times N_{node} \times F\,. \tag{1}$$

In this benchmark study, we used $F = 10$ chunks per node, fixing the number of points in `lon` and `lat` dimensions to 384 and 320, respectively.

The size of the temporal dimension is varied in order to meet the desired total dataset size as defined be (1). For example, a computation with a chunk size of $S_{chunk} = 128$ MB, $N_{node} = 16$ HPC nodes, leads to a total dataset size of 20.48 GB, and the `ds(time, lon, lat)` shape corresponds to `(20834, 384, 320)`. For Hal, the time coordinate contained daily values ranging from 1 January 1980 to the year 2037. On Cheyenne, the time coordinate contained hourly data. The longitude varies from -180 to +180 degrees, and latitude varies from -90 to 90 degrees.

Chunking Scheme Three different chunking schemes were tested:

- The *auto* chunking scheme lets Dask automatically determine the shape of each chunk, given a particular chunk size. The auto chunking scheme subdivides every dimension in order to achieve the desired chink size.
- The *spatial* chunking scheme keeps the temporal dimension contiguous in one chunk, dividing data along spatial dimensions.
- The *temporal* chunking scheme keeps all spatial dimensions contiguous in one chunk, dividing data long the temporal dimension.

With the above Dask dataset example, `ds(20834, 384, 320)`, the auto, spatial and temporal chunking schemes will lead respectively to the following chunk sizes: `(251, 192, 160)`, `(20834, 28, 28)`, and `(131, 384, 320)`.

Geoscience Operations The following four geoscience operations were used to measure performance:

- The **temporal mean** operation is a temporal average. It corresponds to the following code in Xarray:

  ```
  ds.mean(dim='time')
  ```

- The **spatial mean** operation is a spatial (*i.e.*, along `lon` and `lat`) average. On Hal, it corresponds to the following line of code in Xarray:

  ```
  ds.mean(dim=['lat', 'lon'])
  ```

  while on Cheyenne, the spatial mean includes weights.
- The **climatology** operation calculates a standard climatology analysis by calculating the seasonal mean value of `ds(time,lat,lon)`. This operation runs along the time axis. It corresponds to the following lines of code in Xarray:

  ```
  ds_g = ds.groupby('time.season')
  climatology = ds_g.mean(dim='time')
  ```

– The **anomaly** operation computes the anomaly of `ds(time,lat,lon)` with respect to the seasons (*i.e.*, the `climatology` result). It corresponds to the following line of code in Xarray:

```
ds_g - climatology
```

The run time for each operation was measured after the dataset was created and loaded into memory. For each choice of chunk size, chunking scheme, the geoscience operation was performed multiple times, and the median run time for each operation in shown in this paper, reflect the real usage of an typical HPC user.

Strong Scaling Analysis A strong scaling analysis keeps the total size of a problem constant (*i.e.*, the dataset) and evaluates computation times with an increasing number of processes. It allows the problem to possibly scale with the increase of parallel processors and to highlight critical values. Without parallel computing overhead, such as communication or synchronisation, the run time is expected to decrease as $1/N_{node}$. In this study, we fixed the total dataset size to 20.48 GB. The number of nodes N_{node} was varied over 1, 2, 4, 8 and 16, while the chunk size S_{chunk} was varied with the number of nodes from 2.048 GB to 128 MB, such that the total dataset size as defined in (1) stayed constant.

We produced and analyzed 60 sets of benchmark results (four geoscience operations × three chunking schemes × five values for N_{node}). We have performed this benchmark both on the Hal and Cheyenne supercomputers. Using Hal, a total of 1056 computations were performed, with each test set being performed 10 to 28 times. The run times on Hal varied from 1.25 to 77.61 seconds. Using Cheyenne, a total of 96 computations were performed, with each test set being performed one to three times. The run times on Cheyenne varied from 1.10 to 57.39 seconds.

Weak Scaling Analysis A weak scaling analysis aims to determine how the time to solution varies with processor count for a fixed problem size *per processor*. In an ideal case, we expect to observe a constant time to solution, independent of the total number of processors in the system.

In this study, we fixed the chunk size S_{chunk} and varied the total dataset size S_{total} with the number of nodes N_{node}. We performed four different weak scaling analyses using a chunk size S_{chunk} of 64, 128, 256 and 512 MB. For each analysis, the number of nodes N_{node} varied over 1, 2, 4, 8, and 16. We produced and analyzed 240 sets of benchmark results (four geoscience operations × three chunking schemes × five values for N_{node} × four variations of chunk size).

At the time of this publication, the weak scaling study results for Cheyenne are incomplete and are not shown. However, a thorough weak scaling study was performed on Hal. In total, we performed 5268 computations using the Hal supercomputer. Each test set was computed from 20 to 28 times on Hal and from 1 to 2 times on Cheyenne. The run times on Hal varied from 0.49 to 125.22 seconds, and the run times on Cheyenne varied from 0.40 to 91.75 seconds. For

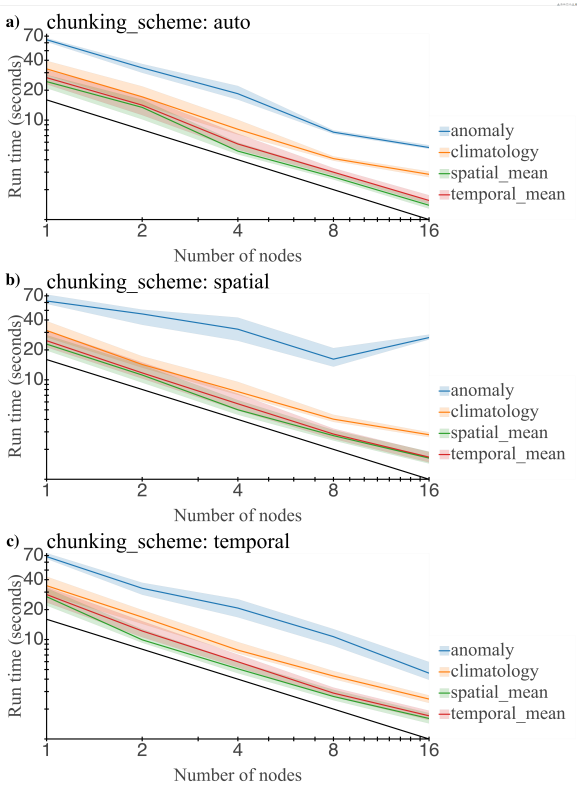

Fig. 1. Strong scaling analysis results for a total dataset size of 20.48 GB using the Hal supercomputer. The x axis shows number of nodes used for each test, shown on log scale. The y axis shows the run time in seconds on a log scale. The blue, orange, green and red lines correspond respectively to the run times for the anomaly, climatology, spatial mean and temporal mean operations. Curves corresponds to the median run time, and the shadowed area shows a single standard deviation from the mean run time. The black line corresponds to the expected strong scaling curve, a^{-1}. From the top, figures a), b) and c) show the run times with auto, spatial and temporal chunking schemes, respectively.

each set of tests, the run time was normalized by the median of non-parallel ($N_{node} = 1$) test.

4.2 Results and Discussions

Strong Scaling Analysis The benchmark results using Hal and Cheyenne are shown in Figures 1 and 2, respectively. For the auto (Figures 1-a and 2-a) and temporal (Figures 1-c and 2-c) chunking schemes, the run time decreases for all four geoscience operations with a a^{-1} power law. This is consistent with the expectation.

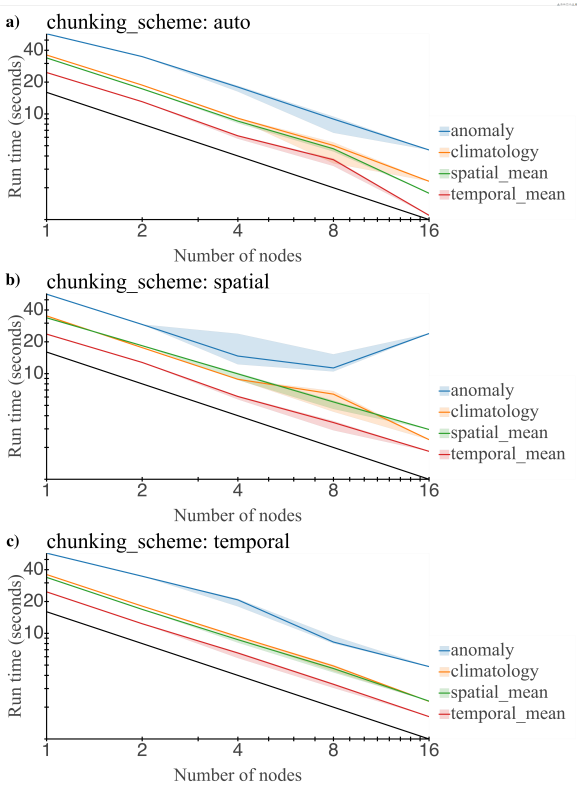

Fig. 2. Strong scaling analysis results for a total dataset size of 20.48 GB using the Cheyenne supercomputer. The x axis shows the number of nodes used for each test, shown on a log scale. The y axis shows the run time in seconds on a log scale. The blue, orange, green and red lines correspond respectively to the run times for the anomaly, climatology, spatial mean and temporal mean operations. Curves correspond to the median run time, and the shadowed area shows a single standard deviation from the mean run time. When only one run was performed, the standard deviation is displayed on plots as zero. The black line corresponds to the expected strong scaling curve, a^{-1}. From the top, figures a), b) and c) show run times with auto, spatial and temporal chunking schemes, respectively.

Dask's automatic parallelism scales well for this cluster size for most chunking schemes. With the spatial chunking scheme, each chunk holds all the data along the time dimension. It is appropriate for operations that run along time (*i.e.*, the temporal mean and climate operations). The anomaly operation also runs along the time coordinate, so we expect it to scale appropriately as well. Run time decreases for the temporal and climatology operations as expected (Figures 1-b and 2-b, red and orange lines). However, the anomaly operation does not scale after 8 nodes (Figures 1-b and 2-b, blue lines.)

We do not fully know why the anomaly operation does not show scaling beyond 8 nodes when using spatial chunking. We suspect that it is due to extra overhead or unnecessary communication or both. Further investigation is planned to understand this problem.

It is clear from the findings that the auto chunking scheme would be a suitable choice for general use cases on HPC.

Weak Scaling Analysis The benchmark results for spatial and auto chunking scheme are shown in Figures 3 and 4 respectively.

None of the operations studied, for either the spatial or auto chunking schemes, show a constant normalized run time as the number of nodes increases. Most operations show a deviation from ideal scaling over 1 to 16 nodes, ranging from roughly 10% to 40% when the idea chunk size is used. However, the anomaly operation, when used with the spatial chunking scheme (Figure 3-a), shows extremely poor scaling. However, the anomaly operation, when used with the auto chunking scheme (Figure 4-a), shows better scaling, though not ideal. These results are consistent with the results in the strong scaling analysis.

For the auto chunking scheme, Figures 4-c and 4-d show that the spatial mean and temporal mean operations scale fairly well regardless of chunk size. However, for the anomaly (Figure 4-a) and climatology (Figure 4-b) operations, a chunk size of between 256 MB and 512 MB scales better compare to other smaller chunk sizes. Larger chunk sizes places more data on each Dask worker, therefore reducing the communication overhead. Dask's default chunk size for the auto chunking scheme is 128 MB. Note that a bigger chunk size requires more memory on the HPC node.

5 Conclusion and Further

The Pangeo community unites scientists and technologists together to make it possible to explore geoscience data using HPC or cloud in an interactive manner. Interactive usage gives a way for researchers to rapidly code and test their ideas [9], but our experiences suggest that it may also introduce some 'blind spots' due to its ease of use. For example, such an easy-to-use parallel platform makes it also easy for users to forget that they are dealing with Terabytes of data with hundreds of workers (*i.e.*, that their machine has real limits and that not all data sizes can easily be analyzed).

This benchmark study of the Pangeo platform shows that the best scalability was obtained with chunk sizes between 256 MB and 512 MB, and, compared to certain manual chunking schemes, the auto chunking scheme scaled well.

Compared to legacy parallel programming models (*e.g.*, MPI), users of Dask do not have to deal with the difficulty of coding their own parallelism. However, they still have to think about grid size and related issues, such as the chunk size and the chunking scheme most appropriate to the computation and the machine they are using. Fortunately, Dask's auto chunking scheme seems to scale quite

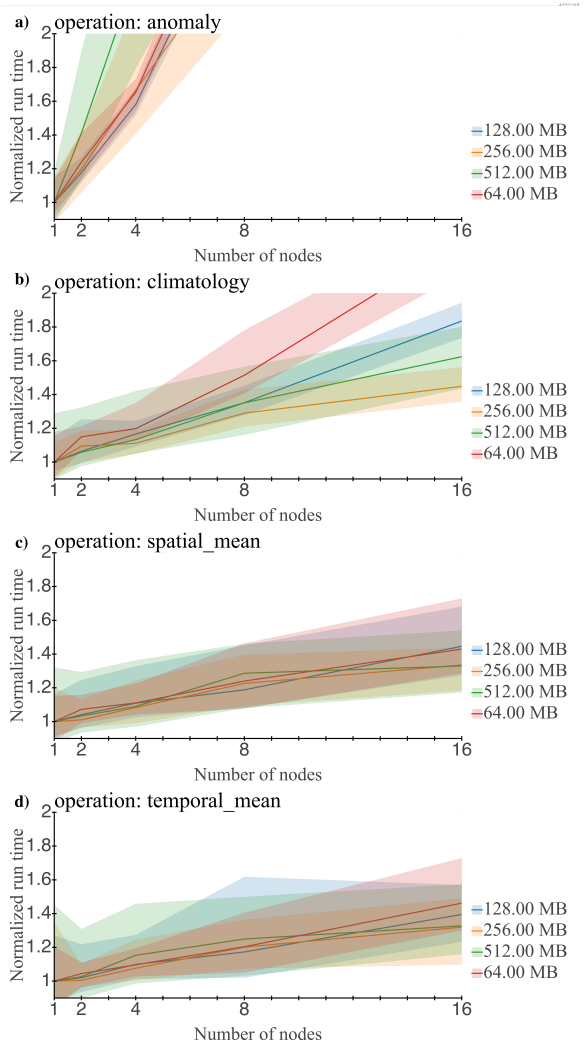

Fig. 3. This figure shows the weak scaling analysis results for the spatial chunking scheme. The x axis shows number of nodes used for each test. The y axis shows the operation run time normalized by the 1-node runtime. The red, blue, orange and green colors correspond to chunk sizes of 64, 128, 256 and 512 MB, respectively. The curves correspond to the medians of run time, and the shadowed areas show a single standard deviation from the mean run time. From the top, figures a), b), c) and d) show the anomaly, climatology spatial mean and temporal mean operations, respectively.

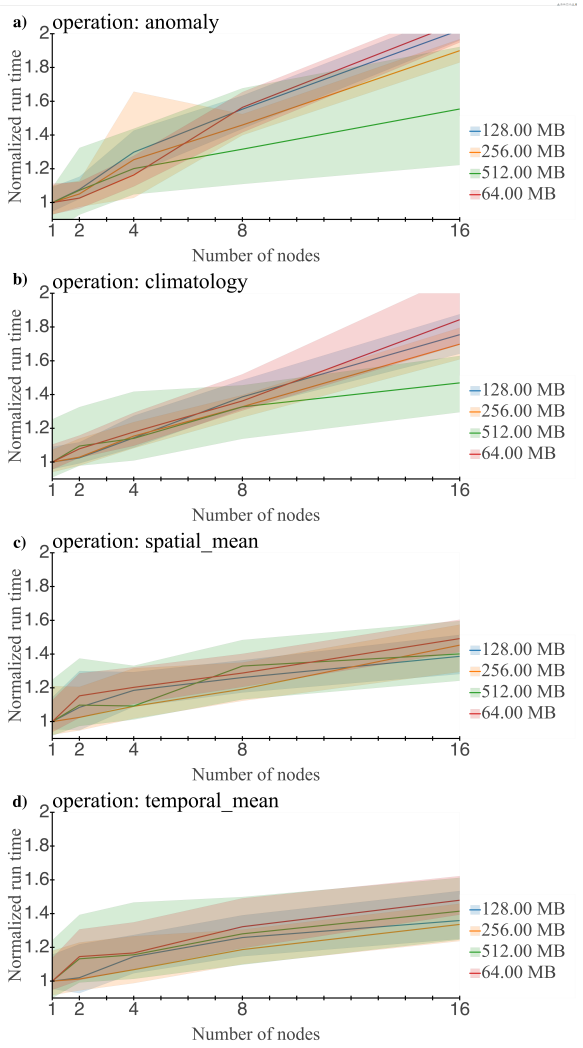

Fig. 4. This figure shows the weak scaling analysis results for the auto chunking scheme. The x axis shows number of nodes used for each test. The y axis shows the run time normalized by the 1-node run time. The red, blue, orange and green colors correspond to chunk sizes of 64, 128, 256 and 512 MB, respectively. The curves correspond to median run times, and the shadowed areas show a single standard deviation from the mean runtime. From the top, figures a), b), c) and d) shows the anomaly, climatology spatial mean and temporal mean operations, respectively.

well, and the knowledge of using a larger-than-default chunk size (*i.e.*, larger than 128 MB) is easy to communicate to users.

The benchmark code used for this paper is open source, and it is published on GitHub [23]. The development of the benchmarking suite continues, with the goal of this benchmarking suite being that user (or administrator of an HPC center) can run these benchmarks and find out what is the best chunk size, chunking scheme, workers per node, and threads per node for a given HPC cluster for geoscience applications. This will help both optimising the usage of the cluster for HPC administrators and optimise the time for HPC users.

Pangeo is still new to HPC platforms. HPC communities have a history of optimisation and parallelism using HPC platforms. For example, there is a history of automatic parallelism methods (*e.g.*, Fortran co-arrays) and the use of RDMA for communication between nodes [24]. These knowledge and specialization from the HPC community may help the development and optimisation of the Pangeo platform.

Acknowledgment

Dr. Abernathey was supported by NSF Earthcube award 1740648. Dr. Paul and Mr. Banihirwe were both supported by NSF Earthcube award 1740633.

References

1. Zender, C.S.: Analysis of self-describing gridded geoscience data with netCDF Operators (NCO). Environmental Modelling & Software **23**(10-11), 1338–1342 (Oct 2008). https://doi.org/10.1016/j.envsoft.2008.03.004
2. Brown, D., Brownrigg, R., Haley, M., Huang, W.: NCAR Command Language (NCL). UCAR/NCAR - Computational and Information Systems Laboratory (CISL) (2012). https://doi.org/10.5065/d6wd3xh5
3. Nitzberg, B., Schopf, J.M., Jones, J.P.: PBS Pro: Grid Computing and Scheduling Attributes. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management: State of the Art and Future Trends, pp. 183–190. International Series in Operations Research & Management Science, Springer US, Boston, MA (2004). https://doi.org/10.1007/978-1-4615-0509-9_13
4. Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.): SLURM: Simple Linux Utility for Resource Management, pp. 44–60. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2003)
5. Community, T.P.: Pangeo: A community platform for big data geoscience. http://pangeo.io
6. Robinson, N.H., Hamman, J., Abernathey, R.: Science needs to rethink how it interacts with big data: Five principles for effective scientific big data systems. arXiv e-prints p. arXiv:1908.03356 (Aug 2019)
7. Guillaume Eynard-Bontemps, Joseph Hamman, A.P.W.R.R.A.: The PANGEO big data ecosystem and its use at CNES. In: Proceedings of 2019 Big Data from Space. pp. 49–52. Munich, Germany (2019). https://doi.org/10.2760/848593

8. Abernathey, R., Paul, K., Hamman, J., Rocklin, M., Lepore, C., Tippett, M., Henderson, N., Seager, R., May, R., Vento, D.D.: Pangeo NSF Earthcube Proposal (2017). https://doi.org/10.6084/m9.figshare.5361094.v1
9. Yu, X., Ponte, A.L., Elipot, S., Menemenlis, D., Zaron, E.D., Abernathey, R.: Surface Kinetic Energy Distributions in the Global Oceans From a High-Resolution Numerical Model and Surface Drifter Observations. Geophysical Research Letters **46**(16), 9757–9766 (Aug 2019). https://doi.org/10.1029/2019GL083074
10. Ponte, A.: Rotary spectral analysis of surface currents and zonal average. https://github.com/apatlpo/mit_equinox/blob/master/hal/rechunk_rotspectra.ipynb
11. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., Jupyter development team: Jupyter Notebooks — a Publishing Format for Reproducible Computational Workflows. In: Loizides, F., Scmidt, B. (eds.) Positioning and Power in Academic Publishing: Players, Agents and Agendas. pp. 87–90. IOS Press (2016). https://doi.org/10.3233/978-1-61499-649-1-87
12. Hoyer, S., Hamman, J.: Xarray: N-D labeled Arrays and Datasets in Python. Journal of Open Research Software **5**(1), 10 (Apr 2017). https://doi.org/10.5334/jors.148
13. Met Office: Iris: A Python library for analysing and visualising meteorological and oceanographic data sets. Exeter, Devon (2010 - 2013), http://scitools.org.uk/iris
14. Matthew Rocklin: Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In: Kathryn Huff, James Bergstra (eds.) Proceedings of the 14th Python in Science Conference. pp. 126 – 132 (2015). https://doi.org/10.25080/Majora-7b98e3ed-013
15. Dask Development Team: Dask: Library for dynamic task scheduling (2016), https://dask.org
16. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache Spark: A Unified Engine for Big Data Processing. Commun. ACM **59**(11), 56–65 (Oct 2016). https://doi.org/10.1145/2934664
17. Hamman, J.: Dask-jobqueue. https://github.com/dask/dask-jobqueue/ (2018)
18. CNES: The Centre National dEtudes Spatiales (CNES) is the government agency responsible for shaping and implementing frances space policy in europe. https://cnes.fr/
19. Computational, Laboratory, I.S.: Cheyenne: SGI ICE XA Cluster (2017). https://doi.org/10.5065/d6rx99hx
20. Ragan-Kelley, B., et al.: JupyterHub — JupyterHub 1.0.0 documentation. https://jupyterhub.readthedocs.io/
21. Willing, C., Zonca, A., et al.: Jupyterhub/wrapspawner. https://github.com/jupyterhub/wrapspawner
22. Milligan, M., Gilbert, M., et al.: Jupyterhub/batchspawner. https://github.com/jupyterhub/batchspawner
23. Paul, K., Banihirwe, A., Odaka, T.: Benchmarking and scaling studies of the pangeo platform. https://github.com/pangeo-data/benchmarking
24. Liu, J., Wu, J., Panda, D.K.: High Performance RDMA-Based MPI Implementation over InfiniBand. International Journal of Parallel Programming **32**(3), 167–198 (Jun 2004). https://doi.org/10.1023/B:IJPP.0000029272.69895.c1