Critique of “Data Flow Lifecycles for Optimizing Workflow Coordination” by SCC Team from Clemson University

Ainara Garcia, Chloe Crozier, Sam Quan, Kristen Guernsey, Benjamin Scheulter, Marcus McAbee

Abstract—This paper reproduces the results of the Supercomputing 2023 paper, *Data Flow Lifecycles for Optimizing Workflow Coordination* by Hyungro Lee et al. as part of the Supercomputing 2023 Student Cluster Competition Reproducibility Challenge. We seek to reproduce the results of the paper titled by implementing the DataLife tool to analyze data flow lifecycles (DFLs) in various distributed scientific workflows. This process involves constructing DFL graphs by measuring and monitoring workflow I/O operations and annotating task dependencies with data properties such as access frequency, reuse, and flow rates. Additionally, we apply DFL-based opportunity analysis to identify and rank potential improvements in data placement, task assignment, and data movement. Experiments are conducted to demonstrate workflow optimizations, including performance enhancements of up to 15x by refining task placement and reducing data movement. We were not able to demonstrate exact performance characteristics from the original paper due to the hardware limitations of our cluster, but we are able to reproduce similar trends considering the configurations we used.

Index Terms—Reproducible Computation, Student Cluster Competition, Replication, High-performance Computing, Distributed Workflows, Datalife, Storage Bottlenecks

1 Introduction

Each year for the Reproducibility Challenge at the Student Cluster Competition (SCC), teams reproduce results from a paper accepted to the previous year’s Supercomputing (SC) Technical Program [1]. For the 2024 reproducibility challenge, teams are tasked to reproduce results from Lee et al. and their SC’23 paper, *Data Flow Lifecycles for Optimizing Workflow Coordination* [2] as part of the SC’24 Student Cluster Competition Reproducibility Challenge. Scientific workflows deployed on distributed computing infrastructures face persistent challenges related to data flow coordination and performance optimization, driven by the increasing scale and complexity of data.

The original study by Lee et al. addresses these issues by introducing data flow lifecycle (DFL) analysis, an approach that extends traditional directed acyclic graph (DAG) representations to include data-centric properties, such as access patterns, volumes, and reuse metrics. This enriched model offers a more comprehensive view of data flow dynamics, enabling the identification of key performance bottlenecks and facilitating targeted optimizations in task placement and data movement.

Their work, implemented using the DataLife tool, demonstrated significant improvements in workflow efficiency across diverse scientific applications, highlighting DFL’s ability to optimize task coordination and reduce data movement. In this study, we seek to replicate and validate the findings of Lee et al., evaluating the DFL analysis and performance gains to confirm the success of their methodology across a broad set of workflows.

In this critique, we:

- Check consistency of reported improvements across setups.
- Determine if the approach scales well for larger workflows.
- Examine if DFL analysis applies broadly to varied scientific workflows.

Our critique is structured as follows: Section 2 outlines the software and hardware configuration we use to obtain our results. Section 3 covers the scripts we use to reproduce results. In Sections **??** and **??**, we present our results for tests. In Section 6, we present visualization results, and lastly, we conclude in Section 7.

2 Experimental Setup

The cluster we use for the competition consists of 4 compute nodes (1 Head Node, 2 CPU, 2 GPU). Initially, we used our PowerEdge 7515 to perform CPU-only workloads, but we were able to include two CPU-only nodes, which are Dell PowerEdge R7625. Now, we are using node 1, (Dell PowerEdge Rx7515), to serve as our head node and does not contribute to our workloads. The new CPU-only nodes allow us to perform computation on a larger distribution of workloads, manage larger datasets, and decrease computation time. The two GPU nodes consist of both CPU and GPU cores, which allow us to increase performance and run a larger range of applications. These nodes contain 128 CPU cores and 4 A-100 SXM4 GPU cores each. The sum total of cores for the entire cluster is 848 CPU cores, 8xA100 GPUs, 8 TB of storage, and 2752 GB of memory.

Tables 1 and 2 show the aforementioned hardware configurations.

2.1 Hardware

	Name	Cores	Memory
Head Node (1)	Dell Rx7515	16	128 GB
Node (2-4)	Dell R7625	192	768 GB
GPU* (5-6)	Dell XE8545	128	4x40GB

	CPU	GPU	Specs
***GPU (x2)**	128 Cores	4x A100	A100 SXM4

Our head, CPU-only, node contains an AMD EPYC 7313P 3.0GHz with 16 cores. Onboard storage includes 128 GB of DDR4 RAM and two 1.6 TB SSDs. Since this node is not as heavily used for computation as the other two, we do not require a faster and more power hungry processor.

The other 3 CPU-only nodes contain two AMD EPYC 9654 2.40GHz, with 192 cores. With the increased number of cores in these nodes, we are able to parallelize jobs and decrease computation time.

The GPU-accelerated nodes contain two AMD 7763 2.45 GHz CPUs with 64 cores and 128 threads. Each node also contains 4 NVIDIA A100 40 GB GPUs. Memory includes 1 TB of DDR4 RAM with 3200 MT/s, and storage includes a 960 GB SATA SSD and a 1.6 TB NVMe SSD.

Given the results from the papers, we determined that the shared/mounted file directory proved to have the best shown and speculated performance for our experiments.

2.2 Software

Our cluster is operating system is Rocky Linux 8.10, since it is widely used in HPC and has a large community support network. Environment Modules and Spack will manage application workflows and modifications. Spack, specifically, will help install the reproducibility challenge software and manage versions of modules used throughout the entirety of the competition. In addition to Spack, Datalife [CITATION] and the Reproducibility-specific software was used.

In HPC, provisioning is when one node (the head) is responsible for managing the configurations of all the compute nodes. It will install the operating system on the compute nodes ”on the fly” based on configuration tables to keep the compute nodes in a consistent state. On a provisioned system, the only thing permanent is the head node. We are using xCAT to provision our cluster, which ensures all the nodes are in the same state. This reduces the probability that configuration differences between nodes impact performance.

To set up the environment for reproducibility, our cluster requires python (version 3.7), networkx (version 2.8.8), plotly, pandas, datalife, cmake, c++ compilers, ipython, numpy, pyaml, python, and matplotlib.

3 Execution and Evaluation

We executed all runs over a single node. We got that data by using the libclient.so, or the shared object file, using ld preload before executing each 1000-genome python script. We logged the runs to a text file, and extracted time stamp data to create the paper’s Figure 6. Additionally, we replaced 1000-genome inputs with our own generated stat files to generate our Figure 1 and table 3.

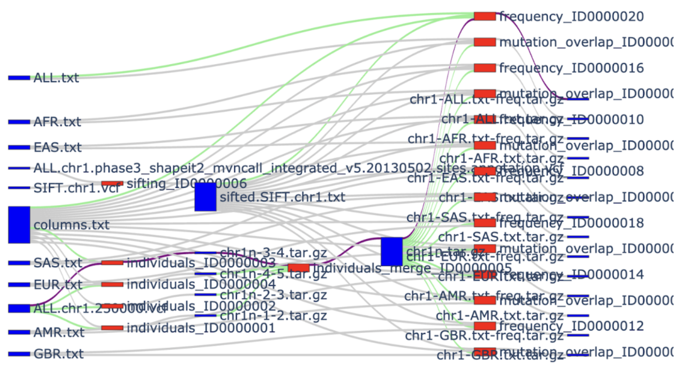


Fig. 1. Dataflow Graph Using Author’s Data

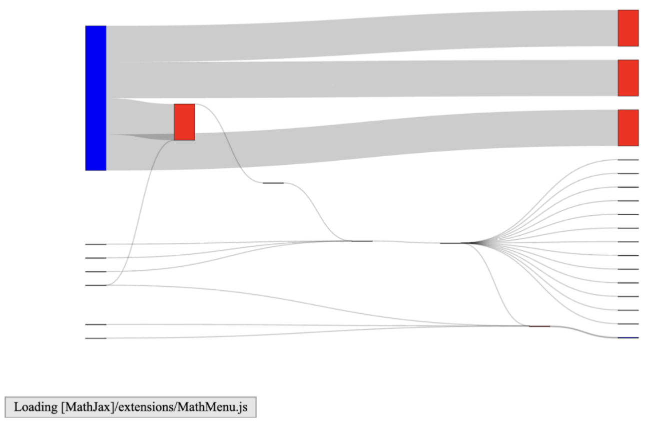


Fig. 2. Caterpillar

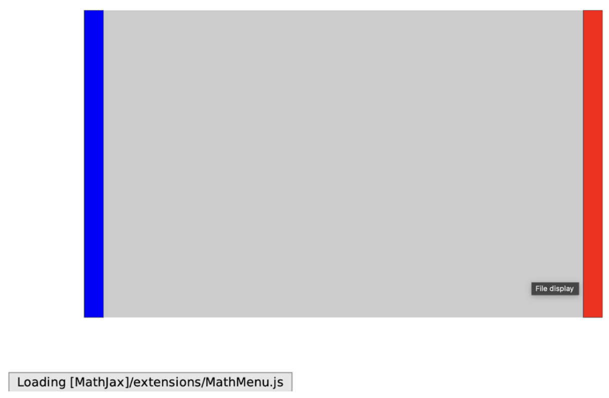
Table 3:
Producer-Data-Consumer

	Volume
individuals-sifting-frequency	105.26
individuals-sifting-mutations	28.71
mutations-sifting-individuals	0.47
frequency-sifting-individuals	0.45

We used the 1k genomes for identifying bottlenecks within workflow runs on various configurations. In the first genomes, the various tasks range from recreating the results from the original paper by running the graph scripts within the Docker container and then installing and compiling datalife to generate our own sets of inputs and compare how our own system behaves to see how our system configurations contributes to the differences in what we see in our results. Figure 2 is a dataflow lifecycle graph showing the access paterns fo reads and writes between tasks. The figure and data were collected on our system, and show the difference between our implementation configuration and the initial paper’s.

Figure 2 shows how our file system is not nested, we are running jobs using the mounted file system leading to dependencies closely connected for straightforward path retreival.

We attempted to identify variations and graph poten-



Loading [MathJax]/extensions/MathMenu.js

Fig. 3. Caterpillar Tree Using CyberTiger's Cluster

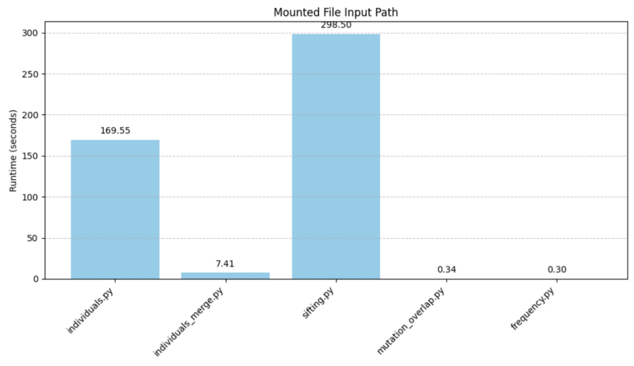


Fig. 4. Mounted File Input Path

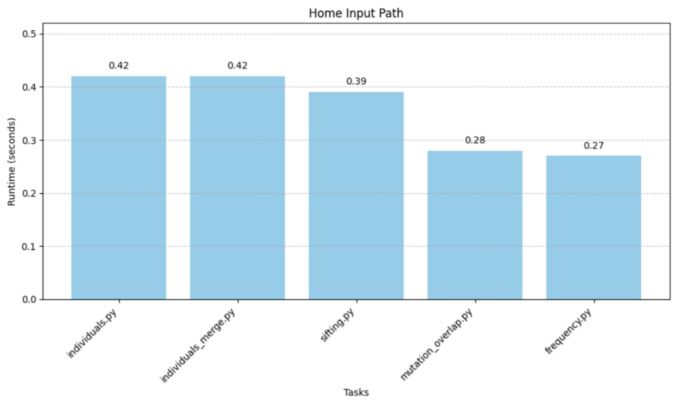


Fig. 5. Enter Caption

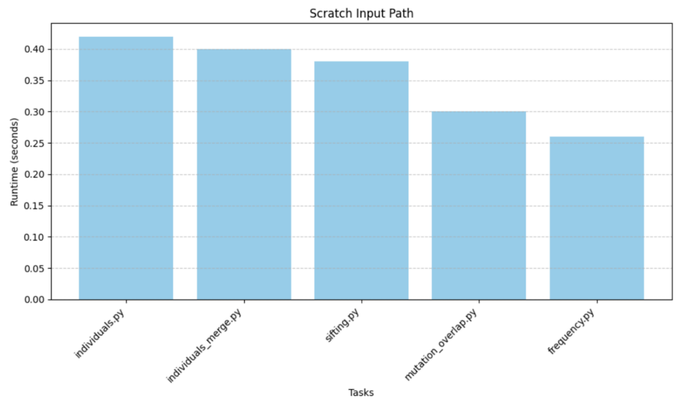


Fig. 6. Enter Caption

tial bottlenecks to decide what kind of things we would see and tried to measure time efficiency of tasks for the 1k genomes for each configuration. Our idea was that if we could identify bottlenecks, for example within the file system, or multitudes of linked problems, then that's where would look at manipulating some of the variables.

Figure 3 is the most accurate. This shows the execution time of the completed workflow on our cluster, based on workflow stage. We encountered many issues when configuring the initial individuals.py staging of our workflow cycle, posing many problems to recreating the experiment with different file system configuration.

The Mounted File Input Path is where we ran the executable in the same directory as the input files, which displayed the best initial performance for parallelization. Figure 4 displays that after staging the input, running individuals and sifting the data took most of the execution time, implying that there is an existing bottleneck when the files transition between these stages. We are able to look back at our Figure 1, Dataflow Graph, to identify any heavy dependencies that might result in a resource or networking bottleneck.

The Home Input Path shows us the potential for limited scalability. Figure 5 looks different than 4 because of the individuals.py's errors in configuration and presumed cache issues for this run. These issues are further explained in the Issues Section. The execution time is must more distributed equally, however we proposes that the lack of scalability and limited resources would show performance when increasing parallelism. Generally we use the /home directory in HPC for temporary, intermediate or small-scaled datasets. This posed a problem with larger datasets like a genome, and could cause further issues with different implementations given more time in the future.

Similarly, using our Scratch as seen in Figure 6 shows the implementation errors and constraints in our system and incompatibility with our tuning potential for our specific configuration. Scratch's role in HPC is to providing high-speed temporary storage for data-intensive jobs, which proved better than /home. This is to be expected as scratch was developed to be temporary and fast-access for parallel computational tasks.

4 Issues

We encountered several challenges in configuring DataLife for our specific cluster setup, particularly during the workflow development phase after staging the input data. Significant issues arose when attempting to run individuals.py to generate the understore stat files, especially when configuring the software for multi-node, multi-process executions.

Adjusting both the problem size and nodal configuration proved to be particularly difficult, as the software did not readily accommodate our cluster's architecture.

To address these challenges, we reached out to the authors and judges of the original paper six times to seek clarification on parameter tuning and guidance for adapting the workflow to our system. Unfortunately, we never received a response to many of our questions, and when we did, the responses were minimal and did not provide enough detail to resolve the configuration issues. As a result, we were unable to achieve a functional multi-node configuration. However, while we could not achieve multi-node execution, we conducted experiments using shared file directories and observed results that supported our hypothesis: shared file directories effectively reduced bottlenecks within workflows and improved performance in parallel computing.

Configuring DataLife for our cluster required substantial modifications and creative, "out-of-the-box" solutions to address incompatibilities. One of the most time-consuming aspects of this process was ensuring that input data was properly staged before running individuals.py. This required several iterations to resolve errors and misconfigurations during both the staging phase and workflow execution. These challenges highlight the complexity of adapting software to diverse architectures and underscore the need for detailed documentation and flexible design to facilitate broader usability.

Though we were unable to get a multi-node run, our one-node, one-task showed that the individual and sifting took the most time during the execution of the workflow. The limiting factor was that individuals.py was unable to be configured for a multi-task run or a multi-node run.

5 Conclusion

Our evaluation highlights the effectiveness of our cluster in managing high-performance computing tasks. By analyzing workflows such as the 1,000 Genomes Project, we identified bottlenecks, particularly in file systems and task dependencies, and assessed task efficiency across various configurations. The high core count of AMD EPYC CPUs and GPU acceleration with NVIDIA A100 GPUs enabled significant parallelization and reduced computation times. Using Rocky Linux 8.10, Spack, and Datalife ensured consistent environments and reproducible results. These evaluations confirm the cluster's ability to handle complex workloads efficiently while providing insights for further optimization and scalability.

Acknowledgments

We would also like to acknowledge the support of Clemson Computing & Information Technology, Dell Technologies, and the SC24 Student Cluster Competition judges.

References

[1] (2023) Student cluster competition. [Online]. Available: https://studentclustercompetition.us/index.html

[2] M. T. J. F. N. R. T. A. K. X.-H. S. Hyungro Lee, Luanzheng Guo, "Data flow lifecycles for optimizing workflow coordination," ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3295500.3356220