Optimizing Workflow Performance by Elucidating Semantic Data Flow

Meng Tang
Illinois Institute of Technology
Chicago, IL, USA
mtang11@hawk.iit.edu

Nathan R. Tallent
Pacific Northwest National Laboratory
Richland, Washington, USA
tallent@pnnl.gov

Anthony Kougkas
Illinois Institute of Technology
Chicago, IL, USA
akougkas@iit.edu

Xian-He Sun
Illinois Institute of Technology
Chicago, IL, USA
sun@iit.edu

1 Extended Abstract

High-Performance Computing (HPC) workflows are becoming more data intensive. For instance, in Sierra [2], MuMMI needs an enormous 154 PB of storage and can reach 1.5 TB/s peak bandwidth [6]. The way data is used in scientific workflows is getting more complicated. Workflows can have multiple stages with different applications doing simulation, analysis, and AI. These applications inter-depend on each other's generated data and can have different workload characteristics, as we see in Montage, DDMD, and 1000Genomic [3, 5, 10]. The current way of transferring data between tasks in HPC is through a shared storage layer like Parallel File System (PFS) and Burst Buffer, but this suffers from problems like slow access and I/O contention [7].

Improving workflow data movement is a challenging task. One can schedule workflow tasks strategically to minimize data movements. Other effective methods, such as data caching and staging, have also shown promise in enhancing I/O performance by reducing computation wait times [11, 12, 14]. With more I/O expertise, one can also utilize and fine-tune various I/O libraries, middleware, and file system configurations. This process often requires iterative profiling and testing. However, these configurations are often optimized for the most general workload, resulting in other workloads still experiencing high latency with shared storage. The persistent challenges revolve around understanding task-data locality and comprehending the data access characteristics of different applications.

Understanding I/O behavior is imperative when deciding on the correct strategies to enhance data access. Details about data access within workflow tasks can effectively guide improvements in I/O and system configuration. Some tools, such as Darshan [4] and Recorder [13], help in profiling and understanding an application's I/O performance. However, there is a lack of tools that analyze how data is accessed across tasks in a workflow and capture semantic information related to its low-level I/O requests. Such tools would be valuable for providing more straightforward insights into data access patterns across multiple tasks. By filling this gap, we can develop better methods for managing data movement and optimizing the overall workflow.

In this work, we unveil a fresh workflow optimization perspective with

(1) careful runtime measurement of data access metrics,
(2) recovering the mapping of domain semantics to low-level I/O operations, and
(3) effective visualization and analysis of semantic data flows for the complete workflow.

2 Methodology

2.1 Hierarchical Data Format version 5 (HDF5)

HDF5 is a widely used storage format in scientific applications. Its hierarchical structure of groups, datasets, and attributes, allows enriched metadata that describes different data characteristics [9]. Additionally, HDF5 library provides extensive API enabling tracking of its high-level data object and the low-level I/O.

2.2 Semantic DAGs and Analysis

Semantic DAGs enhance traditional DAGs by including tasks, filenames, and the **flow of semantic objects** from input files through tasks to output files. Generating Semantic DAGs with performance metrics involves three steps:

(I) Measure data access metrics from a workflow and construct task-data dependency DAG. DAG use task and data vertices, with task execution orders shown from left to right. Sankey diagrams [1] are used to visualize lifecycle graphs.

(II) Two additional layers of vertices are used to show the recovered mapping between HDF5 semantic data and the POSIX-level data location in a file. Data and file vertices are also arranged by last-accessed time.

(III) Performance statistics for data access and I/O are extracted from measurements from both layers. At the semantic object level, where data is accessed in memory, access count and data volume shows the in-memory access stats. On the file level, access count and volume reflect application-to-storage I/O metrics. Bandwidth calculations are indicated by varying lightness of the graph edges.

3 Case Study I: DDMD

DeepDriveMD is a deep learning-driven molecular dynamics simulations workflow for protein folding [**?**]. It consists of a four-stage pipeline including MD simulation and ML-guided sampling.

Figure 1 depicts a single iteration of the 4-stage pipeline, which includes 12 OpenMM simulation tasks, Aggregate, Train, and Inference from left to right. In the OpenMM stage, each task generates

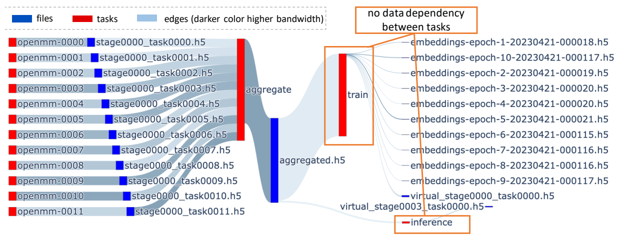
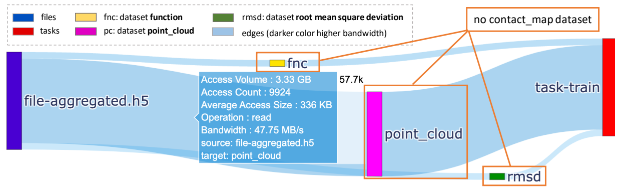

files
tasks
edges (darker color higher bandwidth)
no data dependency between tasks
openmm-0000
openmm-0001
openmm-0002
openmm-0003
openmm-0004
openmm-0005
openmm-0006
openmm-0007
openmm-0008
openmm-0009
openmm-0010
openmm-0011
stage0000_task0000.h5
stage0000_task0001.h5
stage0000_task0002.h5
stage0000_task0003.h5
stage0000_task0004.h5
stage0000_task0005.h5
stage0000_task0006.h5
stage0000_task0007.h5
stage0000_task0008.h5
stage0000_task0009.h5
stage0000_task0010.h5
stage0000_task0011.h5
aggregate
aggregated.h5
train
embeddings-epoch-1-20230421-000018.h5
embeddings-epoch-10-20230421-000117.h5
embeddings-epoch-2-20230421-000019.h5
embeddings-epoch-3-20230421-000020.h5
embeddings-epoch-4-20230421-000020.h5
embeddings-epoch-5-20230421-000021.h5
embeddings-epoch-6-20230421-000115.h5
embeddings-epoch-7-20230421-000116.h5
embeddings-epoch-8-20230421-000116.h5
embeddings-epoch-9-20230421-000117.h5
virtual_stage0000_task0000.h5
virtual_stage0003_task0000.h5
inference

Figure 1: Four-Stages Pipeline Workflow (simulation, aggregate, train, and inference).


files
tasks
fnc: dataset **function**
pc: dataset **point_cloud**
rmsd: dataset **root mean square deviation**
edges (darker color higher bandwidth)
no contact_map dataset
file-aggregated.h5
fnc
57.7k
point_cloud
task-train
rmsd

Access Volume : 3.33 GB
Access Count : 9924
Average Access Size : 336 KB
Operation : read
Bandwidth : 47.75 MB/s
source: file-aggregated.h5
target: point_cloud

Figure 2: DDMD Train Stage Read File I/O Performance Detail.

an HDF5 file, and each file contains four different datasets (*contact_map*, *point_cloud*, *fnc*, *rmsd*). The edges are shown with different lightness of blue, with light edges indicating lower bandwidth and vice versa.

Observation 1: No data dependencies between Train and Inference tasks, as we can see that both of them reads input aggregated.h5, and output different sets of files that are not used by each other.

Opportunities 1: Inference and Train tasks can be parallelized without violating data dependency correctness.

Figure 2 shows Train accessing the aggregated.h5 file, with datasets and performance details. Since each dataset is recorded as HDF5 objects that are in memory, the statistics shown here are memory access size performance. The datasets access order is shown with time progression from left to right on the x-axis.

Observation 2: With a detailed look at the data flow of Aggregate, we can see that it reads and writes the same content but slightly rearranges the logical file address of the two datasets. From figure 3 shown in Poster we know that over 95% of the data volume is from the *contact_map* dataset, while only small amount is from the *point_cloud* dataset. Figure 2 reveal that Train task is not using the dataset *contact_map*. This means the Train task only using a small amount of the data in the aggregated.h5, which further indicates that the aggregate task is not necessary.

Opportunity 2: (1) Since Aggregate is rearranging data without meaningful usage for the downstream tasks, removing this task can reduce unnecessary data movement. (2) Removing Aggregate task may also improve data access parallelism, where data are being accessed from the 12 files generated by each OpenMM task, rather then accessed through a single aggregated.h5 file. (3) When opening up a single HDF5 file, datasets are often prepared into memory before being accessed by the application. And in the case when memory resources become limited in larger experiment workloads, offloading the dataset *contact_map* during Train ensure more efficient memory usage.

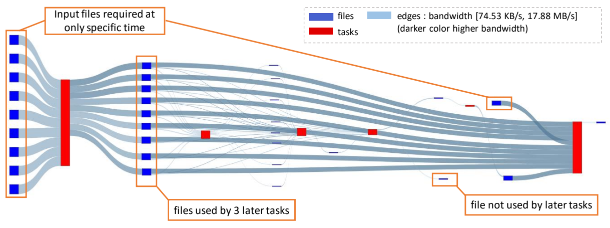

Input files required at only specific time
files
tasks
edges : bandwidth [74.53 KB/s, 17.88 MB/s] (darker color higher bandwidth)
files used by 3 later tasks
file not used by later tasks

Figure 3: Six-Stages Storm Tracking Workflow.

3.1 Case Study II: Storm Tracking Workflow

Storm Tracking uses a flexible atmospheric feature tracking software package [8] for weather research and forecast datasets. The complete workflow typically consists of a 6 to 9 stages sequential pipeline.

Figure 3 shows a six-stage pipeline DAG, where each stage consists of one task, and each stage is executed sequentially. Same as in previous figures, edge color lightness indicates the bandwidth of data access. But time for each file is not presented with the x-axis.

Observation: Here we can see three patterns, (1) inter-task Data Reuse, (2) time-dependent inputs, and (3) data non-used. (1) is shown in tasks 2, 4, and 6. They are reading files generated from the first task, in addition to other files generated by their previous tasks. For (2) we can see from the first set of input files and the top input file for the last task, that not all input files are required at the very beginning of the workflow. We can see that task 4 produces a file that is not used by any later tasks, which corresponds to (3).

Opportunity: There are three opportunities for the above three observations respectively. For (1) we can schedule tasks that use common input datasets on the same compute resources to reduce data movement between tasks. For (2) we can stage-in the inputs at different time points of the workflow execution timeline. For (3) we can offload the data that is not used by later tasks immediately to free up memory when memory becomes a limited resource in the workflow.

4 Conclusion

Nowadays in HPC applications, there is lacks of tools to understand data flow between tasks in a workflow. This study introduced Semantic DAGs, an enriched version of traditional DAGs. Precise measurements allowed us to reconstruct mappings between tasks and meaningful data objects down to low-level file addresses. With careful measurement, reconstruction of data objects to file address mapping, and extracted performance statistics, we can gain new insight for new workflow optimization opportunities.

Our future work will focus on improving the analysis method and developing an automatic approach for intelligent data placement in workflows.

Acknowledgments

This research is supported by the U.S. Department of Energy (DOE) through the Office of Advanced Scientific Computing Research's "Orchestration for Distributed & Data-Intensive Scientific Exploration." Also, the material is based upon work supported by the National Science Foundation under Grant no. NSF CSSI-2104013.

References

[1] [n. d.]. Sankey Diagrams. https://www.data-to-viz.com/graph/sankey.html. Accessed: 2023-03-15.

[2] [n. d.]. Sierra. https://hpc.llnl.gov/hardware/compute-platforms/sierra

[3] GB Berriman, JC Good, AC Laity, A Bergou, J Jacob, DS Katz, E Deelman, C Kesselman, G Singh, M-H Su, et al. 2004. Montage: A grid enabled image mosaic service for the national virtual observatory. In *Astronomical Data Analysis Software and Systems (ADASS) XIII*, Vol. 314. 593.

[4] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)* 7, 3 (2011), 1–26.

[5] Laura Clarke, Xiangqun Zheng-Bradley, Richard Smith, Eugene Kulesha, Chunlin Xiao, Iliana Toneva, Brendan Vaughan, Don Preuss, Rasko Leinonen, Martin Shumway, et al. 2012. The 1000 Genomes Project: data management and community access. *Nature methods* 9, 5 (2012), 459–462.

[6] Francesco Di Natale, Harsh Bhatia, Timothy S Carpenter, Chris Neale, Sara Kokkila-Schumacher, Tomas Oppelstrup, Liam Stanton, Xiaohua Zhang, Shiv Sundram, Thomas RW Scogland, et al. 2019. A massively parallel infrastructure for adaptive multiscale simulations: modeling RAS initiation pathway for cancer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–16.

[7] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, et al. 2011. The international exascale software project roadmap. *The international journal of high performance computing applications* 25, 1 (2011), 3–60.

[8] Zhe Feng, Joseph Hardin, Hannah C Barnes, Jianfeng Li, L Ruby Leung, Adam Varble, and Zhixiao Zhang. 2022. PyFLEXTRKR: a Flexible Feature Tracking Python Software for Convective Cloud Analysis. *EGUsphere* (2022), 1–29.

[9] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 workshop on array databases*. 36–47.

[10] Hyungro Lee, Matteo Turilli, Shantenu Jha, Debsindhu Bhowmik, Heng Ma, and Arvind Ramanathan. 2019. DeepDriveMD: Deep-Learning Driven Adaptive Molecular Simulations for Protein Folding. In *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*. 12–19. https://doi.org/10.1109/DLS49591.2019.00007

[11] Pradeep Subedi, Philip Davis, Shaohua Duan, Scott Klasky, Hemanth Kolla, and Manish Parashar. 2018. Stacker: an autonomic data movement engine for extreme-scale data staging-based in-situ workflows. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 920–930.

[12] Qian Sun, Tong Jin, Melissa Romanus, Hoang Bui, Fan Zhang, Hongfeng Yu, Hemanth Kolla, Scott Klasky, Jacqueline Chen, and Manish Parashar. 2015. Adaptive data placement for staging-based coupled scientific workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[13] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. 2020. Recorder 2.0: Efficient parallel I/O tracing and analysis. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1–8.

[14] Teng Wang, Suren Byna, Bin Dong, and Houjun Tang. 2018. UniviStor: Integrated hierarchical and distributed storage for HPC. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 134–144.