Why wait!? Hades: An Active, Content-Aware System for Precalculating Derived Quantities

JAIME CERNUDA, LUKE LOGAN, ANTHONY KOUGKAS, and XIAN-HE SUN, Illinois Institute of Technology, USA

CCS Concepts: • **Information systems → Record and buffer management**; **Key-value stores**; Main memory engines; *Query operators*; Hierarchical storage management.

Additional Key Words and Phrases: HPC, Data Buffering, query, metadata, derived quantities, hierarchical storage

ACM Reference Format:
Jaime Cernuda, Luke Logan, Anthony Kougkas, and Xian-he Sun. 2018. Why wait!? Hades: An Active, Content-Aware System for Precalculating Derived Quantities. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Modern scientific applications generate vast amounts of data [1]. This data is typically stored using monolithic files on a parallel file system (PFS) [5]. During analysis, applications must process the entire file to derive information, which is highly inefficient due to I/O stalls [4]. To reduce stall times, derived quantities can be pre-computed while the data is produced and then queried in the analysis phase. This approach, however, requires additional capacity and intelligent use of storage hierarchies in order to efficiently store and retrieve derived data.

In this work, we present Hades, an I/O engine that integrates with the Adios2 framework [3]. Hades presents three core benefits to Adios: First, it enhances Adios with a hierarchical buffering of the data generated by scientific applications, allowing for smart data placement and prefetching of data across the entire spectrum of I/O devices available; Second, it is capable of calculating simple derived quantities required by I/O applications, such as the global and local min/max values of a given variable; Third, it presents a smart memory-fist metadata management for querying derived quantities to enhance system performance.

2 ARCHITECTURE

We have developed Hades under the Adios2 framework. Adios works under a put/get I/O interface that Hades needs to adapt to. On put calls, Hades receives the set of data to be stored along with some metadata. Hades initializes two paths. The first is an I/O path that places the data into the hierarchy as individual entities called Blobs. The second is an asynchronous path that operates on the data through user-defined operators to calculate derived quantities. On get calls, Hades receives only the metadata. Hades makes a distinction between two operations: Data calls, asking for the retrieval of data on the system used to support raw extraction of data and Query calls

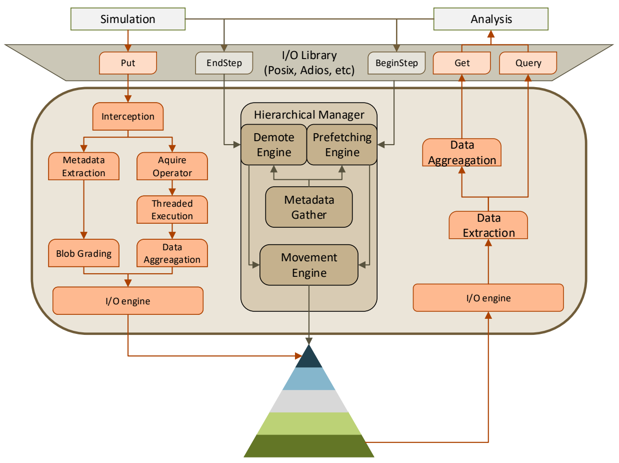

Fig. 1. Hades Architecture

which operate on the derived quantities. These two operations must be treated differently to balance the aforementioned tradeoff between performance and storage use.

Finally, Hades leverages a Hierarchical Manager that leverages the natural step-wise approach of scientific applications to move data through the hierarchy, demoting Blobs to lower layers after being used and promoting Blobs when they are going to be necessary for the next steps of the analysis.

3 HIERARCHICAL MANAGEMENT

To manage the storage hierarchy, Hades leverages two core ideas: First every Blob, a segment of data produced by a process, is assigned a data weight. The data weight takes into account, the blob size, usage frequency, and the number of derived quantities attached to it; Second, through Adios, Hades leverages the natural step-wise design of scientific applications leveraging the *beginStep* and *endStep* calls of Adios.

With these two properties, Hades manages two operations: First, prefetching, which aims to anticipate the data required by the application and promote it to higher layers of the hierarchy before the applications request it. Hades has a parameter, *look ahead steps* that defines how far ahead the prefetcher looks. On a *beginStep* call, Hades will start prefetching the Blobs for the next *n* steps. Blobs with lower data weight are promoted more readily. This technique reduces data access latency and ensures data availability for processing, thereby improving analysis performance; Second, data placement Hades comprises an engine that is responsible for optimal data placement within the hierarchy. Initially, Hades places every data blob in memory. Hades leverage the call to *endStep* in the simulation to demote blobs. Blobs with high data weight are demoted earlier. It uses data placement

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

policies, which aim to maximize application bandwidth and alleviate storage pressure on precious storage resources like memory or NVMe.

4 DERIVED QUANTITIES AND QUERIES

Hades supports calculating on-the-fly derived quantities and the ability to answer simple user queries. We plan to expand on both of these fronts in the future.

Hades currently incorporates two queries: first, the InquireVariable function which is used to obtain information about a specific variable. This method permits collecting important details about a variable without loading all its content into the memory file; second, the MinMax function is responsible for computing the minimum and maximum values of a variable. To calculate it, Hades leverages its derived quantity pathway, where an asynchronous thread calculates the local MinMax on the variable. Once done, it leverages the MPI communicator of Adios to exchange information to process 0, which will formalize the results and place them into a bucket with the result tagging the data based on the variable and step.

5 THE I/O MODEL AND METADATA MANAGEMENT

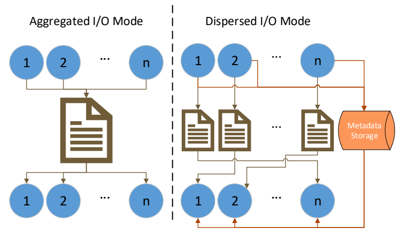

Fig. 2. I/O Model

Hades has two methods of storing variables in the hierarchy: aggregated I/O and dispersed I/O. Aggregated I/O will store variables as a single Blob. This has a low metadata cost and makes variable retrieval simpler. However, this imposes a synchronization step on the write I/O path, which reduces scalability. Dispersed I/O stores variables using a blob-per-process. This improves the scalability of I/O on the write I/O path, but at the expense of increased metadata usage and an aggregation step to read the variable.

When writing a variable, Hades will use dispersed I/O to maximize write performance for the raw data. For the derived quantities, Hades uses aggregated I/O since the operations happen on an asynchronous thread detached from the I/O path, optimizing the future query operations without affecting the main I/O path.

6 PRELIMINARY RESULT

All experiments were conducted on our local cluster, which is designed to support hierarchical storage research. The cluster consists of a compute rack with 32 nodes. The nodes are interconnected by two isolated Ethernet networks (one of 40Gb/s and the other 10Gb/s), with RoCE enabled. Each compute node has a dual Intel(R) Xeon Scalable Silver 4114, 48 GB RAM, and an NVMe PCIe x8 drive.

6.1 Correctness

We run the Gray-Scott [2] Adios application, which consists of a simulation and an analysis step. Combined they make use of both I/O operations (put/get) and queries. We checked for any difference between the output of the BP5 engine of Adios and Hades, both in single and multi-node deployments. We note that Hades provides identical results except when dealing with floating point numbers, and specifically doubles, on query results. This is caused by a difference in the serialization mechanism. Hades cereal-based [6] serialization is more performant, as such Hades allows the user to decide between the two at compilation time, a tradeoff between performance and absolute correctness. Note that the difference in floating point number is on average $3*10^{-8}$.

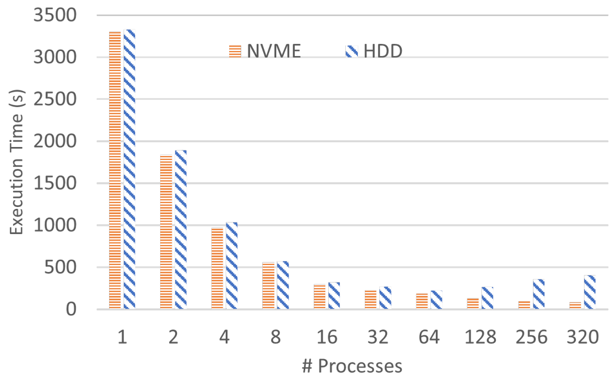

Fig. 3. Simulation Time

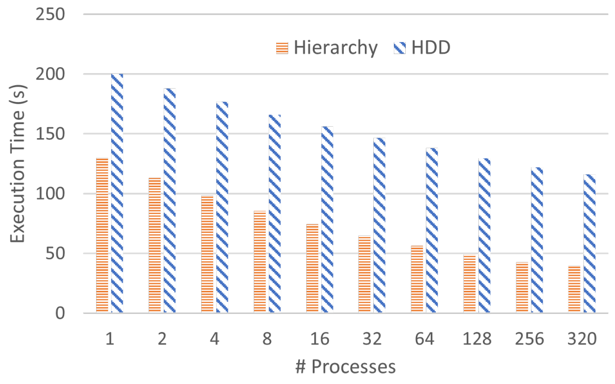

Fig. 4. Analysis Time

6.2 Motivational results

We have executed the Gray-Scott simulation on top of Adios using the BP5 engine. We have both the simulation and analysis components of the application storing the data both in a node-local burst buffer on NVMe and on a remote PFS on HDD.

We can see the benefits that NVMe provides over HDD, especially under a high number of processes. Similarly, the significant overhead that queries impose on the Adios BP5 engine when calling a high number of processes. These results motivate an engine like Hades that can leverage the hierarchy and optimize queries and metadata to achieve better performance on the analysis step.

ACKNOWLEDGMENTS

This work is partially funded by NSF CSSI-2104013 and NSF OCI-1835764.

REFERENCES

[1] Jaime Cernuda, Hariharan Devarajan, Luke Logan, Keith Bateman, Neeraj Rajesh, Jie Ye, Anthony Kougkas, and Xian-He Sun. 2021. Hflow: A dynamic and elastic multi-layered i/o forwarder. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 114–124.

[2] William F Godoy. [n. d.]. *ADIOS2-Examples*. https://github.com/ornladios/ADIOS2-Examples

[3] William F. Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, Axel Huebl, Mark Kim, James Kress, Tahsin Kurc, Qing Liu, Jeremy Logan, Kshitij Mehta, George Ostrouchov, Manish Parashar, Franz Poeschel, David Pugmire, Eric Suchyta, Keichi Takahashi, Nick Thompson, Seiji Tsutsumi, Lipeng Wan, Matthew Wolf, Kesheng Wu, and Scott Klasky. 2020. ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management. *SoftwareX* 12 (2020), 100561. https://doi.org/10.1016/j.softx.2020.100561

[4] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. 2018. Hermes: A Heterogeneous-Aware Multi-Tiered Distributed I/O Buffering System. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (Tempe, Arizona) *(HPDC '18)*. Association for Computing Machinery, New York, NY, USA, 219–230. https://doi.org/10.1145/3208040.3208059

[5] Luke Logan, Jaime Cernuda Garcia, Jay Lofstead, Xian-He Sun, and Anthony Kougkas. 2022. LabStor: AModular and Extensible Platform for Developing High-Performance, Customized I/O Stacks in Userspace. In *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 309–323.

[6] Randolph Voorhies. [n. d.]. *Cereal*. https://github.com/USCiLab/cereal