2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 979-8-3503-9566-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCGrid59990.2024.00070

Hades: A Context-Aware Active Storage Framework for Accelerating Large-Scale Data Analysis

Jaime Cernuda
Illinois Institute Of Techonogy
jcernudagarcia@hawk.iit.edu

Luke Logan
Illinois Institute Of Techonogy
llogan@hawk.iit.edu

Ana Gainaru
Oak Ridge National Laboratory
gainarua@ornl.gov

Scott Klasky
Oak Ridge National Laboratory
klasky@ornl.gov

Jay Lofstead
Sandia National Laboratory
gflofst@sandia.gov

Anthony Kougkas
Illinois Institute Of Techonogy
akougkas@iit.edu

Xian-He Sun
Illinois Institute Of Techonogy
sun@iit.edu

***Abstract*—Modern simulation workflows generate and analyze massive amounts of data using I/O libraries like Adios2 and NetCDF. Although extensive work has optimized the I/O processes during the simulation phase, executing analytical queries—which often require iterative traversals of large files for insights—is cumbersome and usually constrained by low I/O performance. Instead of waiting for the analysis phase to process queries, quantities can be derived asynchronously during data production and cached, speeding up future queries. In this work, we introduce a context-aware I/O layer named 'Hades.' It is designed to efficiently derive insights from selected quantities without compromising overall workflow performance. Hades actively and asynchronously computes and stores these quantities while the data is in transit. Hades leverages a hierarchical buffering system with data access-aware prefetching to ensure quick and timely access to relevant data. It offers a flexible query interface empowering users to easily define derived quantities and provide control over data placement decisions. Hades is implemented using an Adios2 plugin engine and the Hermes buffering platform, enabling transparent use by any Adios-powered application or workflow. Experimental results demonstrate performance improvements by up to 3-4x for tested real-world scientific producer-consumer workflows.**

***Index Terms*—Active Storage, Hierarchical Storage, Context Awareness, Metadata Management, Data Operator, In-transit Computing**

I. Introduction

In high-performance computing (HPC), data-intensive applications have become increasingly common, generating and analyzing vast amounts of data [1]. These applications are bottlenecked by their I/O phases [2], a departure from traditional CPU-bound applications. The high complexity of leveraging I/O to its maximum potential has led to modern applications being dependent on sophisticated I/O libraries, such as HDF5 [3], ADIOS [4], [5], or NetCDF [6]. These I/O stacks are used to manage and execute the I/O required when generating these vast datasets, which are typically stored on a parallel file system (PFS) [7] due to their size.

Significant work has been performed to optimize the write (or simulation) phase of the application [8]. However, the analysis phase in scientific applications continues to face notable I/O bottlenecks. This area remains underdeveloped, with only a limited range of tools available for scientists to explore and derive insights from their simulations. Outside HPC systems, querying systems [9], [10] (usually based on SQL) have existed for decades, allowing users to place their data into databases and execute complex queries to retrieve the specific data required for their analysis. Yet, none of these systems support the data layouts used in scientific computing [3]. In HPC, systems such as Paraview [11], VisIt [12], or the Adios query layer [13], have emerged to aid scientists in this process through specialized querying APIs. Yet, analyzing these datasets to extract valuable features of interest requires heavy and costly I/O operations [14]. During analysis, applications must either comb through the entire file to retrieve their intended data or execute large data movements across the cluster into the compute memory. This imposes significant pressure on the PFS and the metadata servers, utilizes large amounts of expensive in-node memory, and stresses the network's bandwidth. To further exacerbate this problem, a significant amount of the data read is rarely required in its raw form and instead requires passing through a layer of data transformation before the application uses it. This increases the amount of data moment required; for example, having to read three floating point numbers to calculate a single vector norm. Optimizing this process is a requirement to face the increasing amounts of data generation and to reduce the time-to-insights for the scientific community.

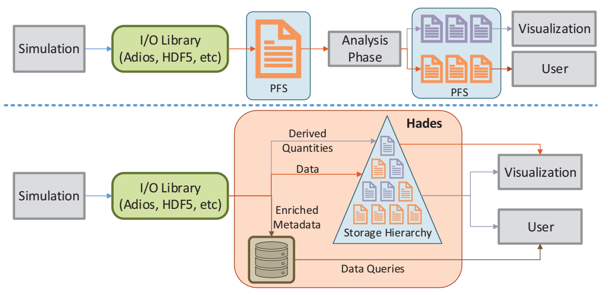


Fig. 1. Hades extends I/O libraries to actively derive quantities of interest (e.g., statistics, integrals) during data production phases allowing for complex queries of the data. Hades reduces workflow complexity, superfluous IO movements and improves time-to-insights.

In this work, we present Hades, an I/O engine capable of calculating derived quantities and serving them to analysis applications. Hades presents scientists an intuitive calculator language to define data-intensive derived quantity expressions useful for analysis. These derived quantities are actively pre-computed while the data is produced and then queried in the analysis phase using a SQL-based API. Since this approach requires additional storage capacity, Hades intelligently leverages storage hierarchies to efficiently store and retrieve derived data. Hades balances this trade-off through the use of a hierarchical buffering engine that is capable of levering all storage devices on the cluster with intelligent prefetching and data placement algorithms. These algorithms can make well-informed placement decisions based on the I/O characteristics and structure provided by higher-level I/O libraries.

This work's contributions are as follows:

1) Enabling the ability to perform insight derivation from data in-transit through a flexible derived quantity calculator accelerating converged scientific workflows.
2) Minimizing I/O stall times caused by derived data generation and data analysis through context-aware hierarchical data placement and prefetching policies.
3) Facilitating insight extraction from both original and derived data with reduced latency using enriched metadata.
4) An open-source implementation of Hades with its design experimentally validated.

II. Background

A. Scientific Inquiry

The escalation of data volumes to exabyte scale poses significant challenges in data analysis. Although raw data is initially stored in monolithic files on the PFS, scientists need elevated information known as derived quantities, requiring the identification, extraction, and transformation of data. However, the growth in data sizes complicates the identification of quantities of interest within extensive datasets. PFSs, while capable of handling large data volumes, fall short in providing efficient, small-sized and selective data retrieval. In addition, exploratory queries often require extensive metadata operations, a known pain point for PFS. Furthermore, the transformation of raw data into meaningful information demands computational resources, leading to delays in insight extraction. Libraries like Paraview [11], VisIt [12], or the Adios query layer [13], help describe and present the data layout and representation. Yet, this systems are design to work offline and cannot compute derived quantities, identify regions of interest or track valuable insights typically performed manually by traversing the original raw data.

B. High-Level I/O Libraries

I/O frameworks have become popular in the HPC community to help alleviate the high complexity of I/O management in the exascale era and to provide newer non-POSIX interfaces. Adios (Adaptable I/O System) [4] is one such I/O framework and is widely used by data-intensive scientific applications and workflows. It has been designed to provide an adaptable interface for data operations, making it capable of portably managing large data volumes. Adios2 benefits are based on three core concepts: The *step*, which refers to an execution phase in the data processing workflow. Each step represents a distinct stage in the lifecycle of data, allowing for incremental and sequential processing. By dividing the workflow into steps, Adios establishes a defined structured to data operations; *variables*, which in Adios are not just data containers but are also associated with contextual metadata such as dimensions, type, and application-specific attributes.

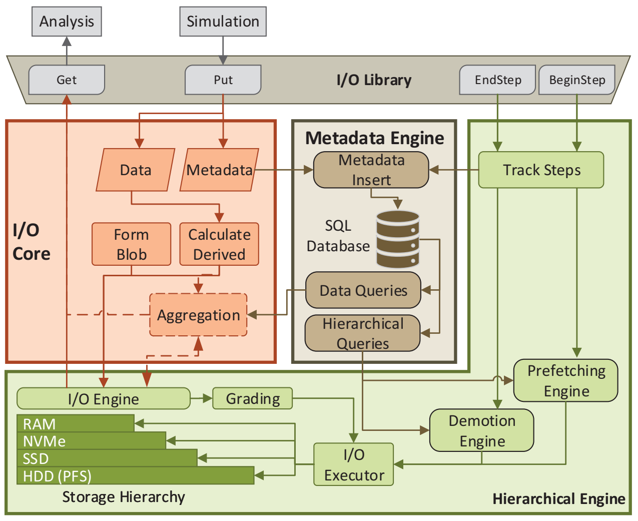


Fig. 2. Hades Architecture

Variables can represent simple scalars or complex multidimensional arrays; *engines*, libraries or modules within Adios implementing its interface. By employing different engines, Adios offers adaptability and optimization for various data operations and data formats. Hades is currently implement as an engine.

C. Hierarchical Data Management

HPC clusters have adapted to the data growth by integrating additional storage tiers, including node-local and remote burst buffers. While these adaptations enhance storage capacity and performance, they introduce deep hierarchies that are challenging to optimize and manage [15]. For this reason, hierarchical data management systems have emerged to simplify and automatically optimize the data storage and retrieval process across the spectrum of available I/O devices. State-of-the-art systems [15]–[17], can allocate data to any layer of the hierarchy transparently. This allows user data to be stored in a range of locations, from high-speed, low-capacity tiers to slower, high-capacity ones. In this work, Hermes [15] is our hierarchical data manager of choice. It is fully released and presents a Put/Get API. Additionally, Hermes presents a factory interface to its data placement and prefetching algorithms, which by default is an LFU-based algorithm, allowing scientist to develop new mechanisms to drive this processes.

This difficulty for scientist to extract insights from a complex data and storage space sets the stage for the need to calculate derived quantities in-transit and a better mechanism to extract insights from datasets.

III. Design

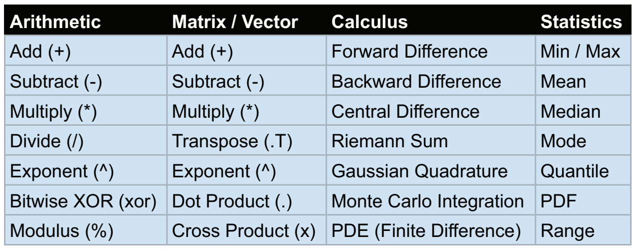
Hades is an I/O engine designed to provide scientists an avenue for actively precomputing and storing quantities of interest to reduce data movements in complex scientific workflows. Scientists define the quantities to derive using a novel high-level calculator language (HDCalc) that provides various mathematical operators and functions, such as aggregation and filtering. Hades asynchronously calculates derived quantities while data is produced, avoiding compute and I/O penalties on the critical path. Hades stores application data and derived quantities in the storage hierarchy using workload-specific data placement engines. Data relevant to future scientific queries will be prefetched and placed in faster storage tiers, accelerating query performance. Hades is developed as a plugin to the widely-used Adios2 I/O library along with an intuitive API extension for querying derived quantities, requiring minimal application change to leverage. The high-level architecture of Hades is depicted in Figure 2. Hades is designed with the following objectives in mind:

1) **Provide semantic derived quantity expressions**: Users should be able to define the quantities to derive using a human-readable mathematical language, avoiding the burden of developing full analytical applications.
2) **Enable efficient and flexible insight extraction**: Users should be able to run complex queries on data and derived data to easily gain meaningful insights.
3) **Leverage intelligent hierarchical management**: Data and derived data should be intelligently placed in the storage hierarchy to minimize data movement costs for derived quantity calculation and querying.
4) **Mediate resource contention**: The calculation and storage of derived data should not cause significant slowdowns to producer phases due to CPU and storage resource contention.

A. A Semantic Language for Defining Derived Quantities

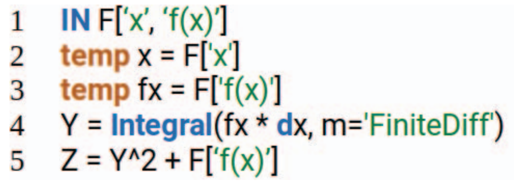
Modern scientific workflows are oftentimes divided into separate phases for data production and analysis [18], [19], where the analysis must repeatedly reload all data that was produced. This distinct separation of phases reduces time-to-insight as it requires significant data drilling and I/O stalls. Hades transitions from this monolithic producer-consumer approach towards an active storage design which produces and analyzes data simultaneously. To enable this live analysis, Hades introduces a high-level calculator language, HDCalc, which provides users the ability to define custom derived quantities using a human-readable mathematical notation. These quantities can then be queried during the analysis and visualization phases with minimal data movement and computational overhead.

HDCalc: a high-level mathematical language: HDCalc provides various built-in mathematical operators, shown in Figure 3. Operators can include traditional arithmetic, aggregation, filtering, integrals, derivatives, partial differentials, statistics, and more. Operators are combined to form equations, potentially consisting of multiple variables and constants. An example equation is shown in figure 4, which integrates a function f regarding x and stores the result in Y. f and x are stored in row-major order in a matrix and passed to Hades by the application. The result of the equation will be automatically stored by Hades as a new variable, which can be used in future equations. By default, the storage duration of variables is persistent (i.e., the data will be stored by Hades and remembered for analysis). Variables can also be marked as temporary, allowing their space to be freed when dependent equations have finished utilizing them. HDCalc does not require users to specify data types explicitly. This is because the data type of a variable is provided to Hades automatically by a higher-level I/O library, such as Adios2. The data types of variables and constants are numeric, including integer, float, vector, and matrix. Lastly, macros can be used to define equations that are used more than once to avoid code repetition.



Arithmetic	Matrix / Vector	Calculus	Statistics
Add (+)	Add (+)	Forward Difference	Min / Max
Subtract (-)	Subtract (-)	Backward Difference	Mean
Multiply (*)	Multiply (*)	Central Difference	Median
Divide (/)	Transpose (.T)	Riemann Sum	Mode
Exponent (^)	Exponent (^)	Gaussian Quadrature	Quantile
Bitwise XOR (xor)	Dot Product (.)	Monte Carlo Integration	PDF
Modulus (%)	Cross Product (x)	PDE (Finite Difference)	Range

Fig. 3. An overview of candidate operators in Hades



```
IN F['x', 'f(x)']
temp x = F['x']
temp fx = F['f(x)']
Y = Integral(fx * dx, m='FiniteDiff')
Z = Y^2 + F['f(x)']
```

Fig. 4. An example of HDCalc. An application produces a matrix F with columns x and f(x). A derived quantity Y is calculated as an integral of f(x) regarding x. Another quantity Z is calculated as the square of Y + f(x).

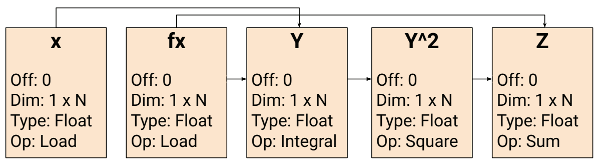


Fig. 5. A depiction of how the equation in Figure 4 is compiled into an **OpGraph**. Vertices represent calculation tasks, which encompass the operations to perform on input data, the name of the output variable, and all metadata required to run the operation. Edges represent the data dependencies between operators. Operators cannot execute until all incoming edges are computed. Vertices store various metadata, including the offset to begin I/O, dimensions, data type, and the operation being performed on the inputs.

Compiling HDCalc into an executable OpGraph: After creating the HDCalc file, it can be passed to Hades using an environment variable. HDCalc is parsed and compiled dynamically by Hades to minimize data movements and calculations. The runtime uses high-level math parsing libraries, such as SymPy [20] and muparser [21], to produce an optimized parse tree. The parsing phase reduces arithmetic, aggregates constants, substitutes macros, and identifies equations that can be executed concurrently. The parse tree is compiled into a low-level schema language, which is then converted into an operation graph (**OpGraph**). The vertices of the OpGraph represent the operations performed on input data, the name of the output variable, and all metadata required to run the operation. Edges represent the data dependencies between variables. Figure 5 shows an example of the task decomposition of the equation in Figure 4. An outgoing edge from variable fx to Z indicates fx is required for Z to be derived.

Use Case: The Gray-Scott Workflow: Gray Scott [18] is a computational fluid dynamics code that models the reaction between chemicals. The model takes as input two components: U and V. U represents the concentration of a substance that promotes uniformity or inhibits the formation of patterns. V

App Name	Variable Name	Number process	Constant Dimensions	Type	Dimensions
App1	VarA	4	True	int	[4,4]
App2	VarB	2	False	char	[2,2,2]

Fig. 6. GlobalTable structure with example entries.

Start	Count	Blob Name	Step	Variable Name
0	4	blob1	1	VarA
4	4	blob2	2	VarA

Fig. 7. Metadata table structure with example entries.

represents the concentration of a substance that promotes pattern formation or acts as an activator. The model measures how U and V change over time and space. One quantity calculated during the analysis phase of this code is the probability density function (PDF) of U and V. This is a data-intensive operation that iterates over the dataset numerous times. As opposed to waiting until the analysis phase, the PDF function of HDCalc can be used to precompute this quantity before the analysis begins.

B. A Flexible Insight Extraction System

Scientific data queries are frequently conducted by manually iterating over massive datasets and manually filtering for information [22]. This is a consequence of I/O libraries and storage systems that do not currently provide APIs to locate subsets of data based on complex mathematical constraints [13]. This manual and synchronous calculation of data-intensive operations leads to increased development complexity and significant performance degradation caused by inefficient data movements. Hades empowers scientific discovery by expanding I/O libraries to support context-aware, user-driven derived metadata, which can be used to identify interesting subsets of data and derived data. By enriching the metadata stored by I/O libraries to support queries over derived data, powerful insights can be gained from complex scientific information while bypassing significant computational and I/O slowdowns.
Leveraging enriched metadata to achieve contextualization: Some of the practical goals of Hades are to calculate derived quantities, manage the hierarchy, perform optimal I/O movement and enable users to extract insights from their data. To achieve these goals, Hades creates, maintains, and uses enriched metadata through a number of metadata structures with various forms and objectives.

Operational Metadata: metadata used in the lifetime of natural Hades execution without user interaction. As shown in Figure 6 this includes variable metadata including variable names, data types, shapes, per-process data distributions, etc. Figure 7 shows another example in which Hades is tracking the specific content of the blobs in the hierarchy. It does so by tracking the concrete offset that the variable represents with respect to the global variable of that a given step. This metadata is queried by Hades and operators to contextualize the data attached to a specific variable (identified by name) which is usually presented as a raw buffer. This information can allow Hades to: transform the raw data into a proper and transformable data type, for example, a vector; or locate the location of the segment of the data buffer required by a specific operator, allowing concrete data movements. A very important use case for operational metadata is ensuring correct data management when the number of producers and consumers does not match, making Hades need to redistribute the data.

Enriched Metadata: metadata attached to data or derived quantities used to elevate the querying capabilities of users. Enriched Metadata can be divided into two categories, inherent or external. Inherent metadata is enriched metadata that can be captured without user intervention, the local and global maxima/minima of a variable or derived quantity for a given step, data ranges, timestamping, etc; More interesting, external metadata is enriched metadata that requires some level of user input to establish a label, a metadata entry on a table similar to Figure 7 characterizing the data point. For example: tagging all points/particles with a property above a threshold, categorizing the data into the generated units (m, C, etc), establishing bounding boxes including all elements with a certain property, etc. Hades provides users the ability to define this tagging operations is derived from a user-defined labeling process which can be defined through an operator on an OpGraph and individually customized per run through a yaml file.
Enhancing scientific inquiry with a query API: Hades asynchronously calculates complex aggregations of scientific data to produce insightful derived quantities. These quantities can be queried using the *InquireVariable* method. This method can be used to query simple metadata of a variable. Yet, more complex queries, such as ranged data queries, must also bee suported, allowing users to load subsets of a variable or derived quantity that meets, for example, certain ranges using mathematical constraints, such as less than and greater than, or falls under a bounding box. Hades uses OpGraphs to execute these data retrieval operators. These operators have access to both the enriched metadata, used to select the desired data, and the operation metadata, used to efficiently extract the data from the hierarchy. A user-defined graph can be submitted to Hades through the query API. Patterns generated by this queries are passed through the hierarchical layer, allowing prefetching of data when queries are repeated over a number of steps, a common IO pattern. When needed, all queries will wait until the subset of metadata and data being queried has been fully derived by Hades.
Use Case: Soil temperature: As an example, the Weather Forecast Model (WRF) [19] produces and analyzes a dataset containing various atmospheric and environmental properties over time and space, such as soil and air temperature. Analysis of these variables is of interest to various domains, including agriculture, ecology, hydrology, civil engineering, and climate science. Enriched metadata can be created to mark data points above a certain average temperature range (between for example soil and air). Queries can be performed to identify the spatial bounding box where these locations exist. Making use the operational metadata to physically locate and retrieve the data where, for example, temperatures are expected to be dangerous for plant life.

C. Leveraging Context to Inform Accelerated Storage

Data placement decisions are typically made without high-level knowledge of the I/O characteristics and behaviors of scientific workflows [15], [23]. This results in suboptimal data movements, which exacerbate I/O overheads. Hades leverages knowledge of data dependencies of derived quantities supplied by the user in addition to hints supplied by higher-level I/O libraries (i.e., Adios2) to inform data movement decisions. To implement these decisions, Hades uses a hierarchical engine, Hermes [15], and enhances it with dynamic policies for data placement and prefetching to adapt to the specific workload.

Using I/O libraries to inform data movement: High-level I/O libraries have numerous characteristics that can help guide data placement. Adios2, for example, requires applications to structure their code in a step-wise, checkpoint-restart fashion. Each process begins a step using the BeginStep function, which either produces or loads a blob of data independently. Typically, steps are followed by a phase of computation where no I/O is happening. When the application finishes using the data, it calls EndStep to free up resources. To optimize data production, Hades will initially place per-process data objects independently in high-performance tiers (e.g., memory and burst buffers) and begin calculating derived quantities relying on this data asynchronously. During data analysis phases, blobs are accessed using a predictable naming convention, which is typically a function of the current step number. Hades leverages this knowledge to prefetch data during BeginStep that will be accessed in the near future. During BeginStep, Hades will perform a look-ahead operation that will promote data based on the next expected access time. This access time is estimated by measuring the average time difference between BeginStep and EndStep throughout the application run.

Decomposing equations into task graphs: In addition to placing the raw data produced by the application, servicing data-intensive equations requires intelligent management of storage, memory, and compute resources. There are two main objectives: ensuring that the equations finish executing before the analysis phase queries them, and ensuring that the data production phase is not significantly hampered by the increased resource consumption from live calculation of data. To execute equations, Hades first converts the equation into an OpGraph, where tasks operate over different subsets of input data. Figure 5 shows an example of the task decomposition of the equation in Figure 4. An outgoing edge from variable fx to Z indicates that fx is required for Z to be derived. The data subsets used as inputs to tasks are determined by the mathematical properties of the operator and the amount of memory, storage, and CPU available for active computation. For example, a matrix addition operator will divide the input matrices into several evenly sized subsets which are processed and stored independently. To address resource utilization, users can configure Hades to bound the maximum amount of memory used for tasks and the maximum number of workers used to execute tasks. Tasks will be scheduled for execution when all input data for the task are available. The Hades runtime will prioritize the execution of tasks where input data is staged in high-performance tiers to minimize data transfer costs and to potentially free up precious storage resources for use by other tasks.

Scheduling data movements to optimize task processing: To maximize I/O performance, data required for tasks to complete will be staged in faster tiers of the storage hierarchy. Data derived by the task will typically be stored with a lower priority than simulation data to minimize data stalls to the main application. This is because derived data is not typically accessed until a distant consumer phase, so polluting buffering space can cause unnecessary slowdowns for the producer. If the derived data is used as an input to another equation, then it will be stored with a higher priority. Data which are no longer needed for the computation of tasks will be moved to higher-capacity storage tiers to free up space for handling bursts from the producer application.

Representing hierarchical data using blobs: Hades stores application and derived variables as blobs. A blob is the combination of a pointer to the data, the size of the data, and various statistics. Blobs are associated with a score between 0 and 1. Higher scores will be placed in faster tiers of the storage hierarchy. Hermes, our hierarchical engine, sets all blobs to 1 and decreases their score monotonically with time. In Hades, scores are determined based on various statistics, including the next expected access, access frequency, access recency, data size, and the number of derived quantity calculations depending on the data. Based on this score and capacity constraints, Hermes will reorganize the blobs asynchronously across storage levels.

Algorithm 1: Calculation of Blob Score B_{Score}

```
Data: Blob variables and parameters
Result: $B_{Score}$
Input  : Variable $B$, task graph $G$, prefetch score $B_{Pre}$,
         Time window for accesses to be recent $T_{Recent}$,
         Hades importance score $B_{Hades}$, configurable
         weights $W_{Task}$, $W_{Hot}$, $W_{Pre}$, and $W_{Hades}$
Output : Blob score $B_{Score}$
1 $B_{Task} \leftarrow G$.OutDegree($B$) / $G$.MaxOutDegree();
2 $B_{Count} \leftarrow$ # of times blob $B$ was accessed;
3 $B_{AvgCount} \leftarrow$ Avg # of times blobs accessed in $T_{Recent}$;
4 $B_{Access} \leftarrow$ Timestamp of the last access to blob $B$;
5 $B_{Hot} \leftarrow B_{Count}/B_{AvgCount} \times \left(1 - \frac{B_{Access}}{T_{Recent}}\right)$;
6 $B_{Score} \leftarrow$
      $\max(W_{Task}B_{Task}, W_{Hot}B_{Hot}, W_{Pre}B_{Pre}, W_{Hades}B_{Hades})$;
7 return $B_{Score}$
```

Blob Scoring: The blob score B_{Score} of a variable B aims to ensure that blobs accessed frequently or in the near future are prioritized for placement in high-performance tiers. The blob score is the foundation of how Hades decides the organization of data in the storage hierarchy. The algorithm is shown in Equation 1. First, the blob task score B_{Task} is calculated, which represents the relative number of active tasks depending on the blob. The intuition is that the more derived quantity tasks depend on the blob, the more it will be accessed in the near future. It is calculated by dividing the outdegree of the variable B in the task graph G by the maximum outdegree.

The blob hotness B_{Hot} is calculated as a function of access frequency and access recency. It aims to prioritize data which is frequently accessed. A configurable time window T_{Recent} (in seconds) is provided where blob reads and writes are considered recent. The number of times the blob was accessed B_{Count} (either read or modified), the average number of times blobs in the hierarchy are accessed per-window $B_{AvgCount}$, and the timestamp of the last access B_{Access}. When the window ends, the blob hotness B_{Hot} will be multiplied by the relative difference between last access B_{Access} and window size T_{Recent}, where B_{Access} is always smaller than T_{Recent}.

The prefetch score B_{Pre} represents how near in the future the blob will be accessed, and is supplied externally by the prefetcher. A higher score indicates the blob will be accessed soon. Lastly, Hades can assign a custom importance score B_{Hades} to represent additional factors of the specific I/O library interacting with Hades. For example, Hades will assign metadata objects a score of 1 to ensure they are always buffered in high-capacity tiers. The final B_{Score} is calculated as the maximum of the $W_{Task}B_{Task}$, $W_{Hot}B_{Hot}$, $W_{Pre}B_{Pre}$, and $W_{Hades}B_{Hades}$ to ensure that blobs ranked important by at least one of these metrics are positioned in high-performing tiers. Weights are configurable numbers between 0 and 1. By default, all weights except W_{Task} are equal to 1. W_{Task} is .7 by default, since derived quantities are calculated asynchronously in the background during data production and computation phases. Setting this score lower than 1 reduces storage contention between data produced by the application and data analyzed and produced by derived quantities.
Initial Data Placement: When initially placing a blob, the blob score is based solely on B_{Task} and B_{Hades}, as this is the only information known about the blob at this time. B_{Hades} will be set to 1 if the blob is metadata related to Hades or the I/O library, and 0 otherwise. B_{Task} will be set based on the task graph. Data will be placed in the fastest storage tier with available capacity. The blob reorganizer will asynchronously promote and demote blobs based on score after initial placement.
Context-Aware Blob Prefetching: The prefetcher aims to anticipate the time at which a blob will be next accessed based on characteristics of the higher-level I/O library. The prefetcher leverages the current step number provided by Adios2 to determine the next blobs to prefetch. Hades measures the average time of the computation phase C_{Avg} by calculating the difference between BEGINSTEP and ENDSTEP functions for each step. Hades will then prefetch the next n steps until $C_{Avg} * n$ is less than the user-configurable window size T_{Recent}.
Dynamic Blob Reorganization: As the application runs, the I/O requirements of the workflow and the available storage capacity will change dynamically. To adapt to these changes, the demotion engine is responsible for correcting the placement of data in the storage hierarchy by asynchronously promoting and demoting blobs based on the score of the blob, the relative score of blobs within a tier, and the remaining capacity of a tier. Periodically, the demotion engine calculates the 0, 25, 50, 75, and 100 percentile of blob scores in each tier. A blob will be targeted for promotion if the blob score is larger than the 25 percentile of the tier. If there is space, the blob will be moved immediately. Otherwise, the blobs in the candidate tier with a lower score will be evicted until there is enough space. If there is still not enough space, the blob will remain in place.

IV. EVALUATIONS

A. Methodology

hardware: All experiments were conducted on our local research cluster, 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 and 48 GB RAM. For storage, each node is equipped with a NVMe PCIe x8 drive, a SATA SSD and a SATA HDD.

The operating system of the cluster is Ubuntu 22.04. All the tests in the evaluations were performed 3 times and the average result is reported. Unless noted otherwise, all data volumes are larger than the allotted memory space. Adios2 is used to test all competing evaluations. Adios2 is always configured to use BP5, TwoLevelShm aggregation, and default configuration for all tests. The parallel file system of choice is OrangeFS v2.9.8.
Software: Hades was implemented in 8K lines of C/C++ code and is publicly available on GitHub[1]. Hades is built under Adios2 v2.9.0 as a Plugin Engine and makes use of Hermes 1.1 to access the hierarchy. Adios2 is enabled with MPICH v4.1.1. For communication, Hades uses Mochi Thallium v0.10.1 with libfabric v1.18.0 and Argobots v1.1 under it. The metadata is stored on SQLite v3.40.1. Currently, Hades implements only simple arithmetic operators, filtering, and statistics.

B. Experimental Results

Our evaluations have the following objectives:

1) Evaluating the integrity and performance of our designs by comparing Gray-Scott simulation outputs with and without the integration of Hades.
2) Demonstrating the I/O performance improvements in scientific workflows using Hades' intelligent management of deep memory and storage hierarchies (Figure 8).
3) Assessing the performance and analytical advantages of Hades' SQL-based metadata management in HPC environments. (Figure 9).
4) Evaluating the impact of Hades on simulation and analysis phases by actively computing derived quantities during data production (Figure 10).
5) Measuring the end-to-end performance enhancement in scientific workflows using Hades for active derivation and hierarchical storage of data during production (Figure 11).

*1) **Operational Correctness**:* Hades offers the ability to offset data-intensive calculations from the application and bring it near the data. This can offer great performance benefits. However, by changing the underlying I/O system, the validity of the I/O can change and the correctness of the application becomes a concern. We demonstrate the validity of our system by running the publicly available [24] Gray-Scott workflow [18] and comparing the simulation outputs with and without Hades. As mentioned in the design, Hades includes an inline profiling engine that leverages spdlog, a fast and performant C++ logging library, to track (if desired) all the variable data and metadata flowing through the engine. Combined with an `LD_PRELOAD`able Adios2 profiling library, Hades is able to present a public data set that records all values of the application when executing with or without Hades. Hades implements and compares almost all of the Adios2 interfaces (except the very recent Adios2 span variables). This includes Adios2 metadata operations (*InquireVariable()*), Put and Gets, and the min-max interface of Adios2. All these interfaces are leveraged by at least one side of the Gray-Scott workflow and are present in the logs, which are available in the public repository of the project.

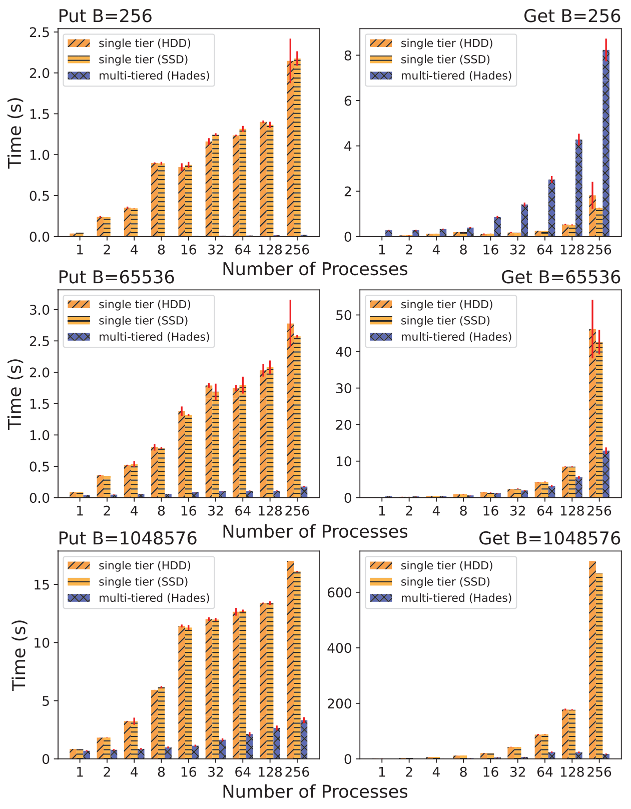


Fig. 8. The performance benefits of leveraging storage hierarchies to buffer workflow stages

2) *I/O Scalability*: Hades leverages the deep memory and storage hierarchy to accelerate the I/O operations of scientific workflows and to alleviate the increased storage demand generated by the calculation of derived quantities. This evaluation aims to showcase the I/O performance benefits that the intelligent management of the hierarchy can bring to scientific applications. This management, which is driven by our data placement and prefetching engine, allows applications to perform most of their I/O in high-performance storage tiers.

For this evaluation, we performed weak scaling on a synthetic Adios2 application. The application initializes a global 2D char array of size *(num_processes, B)*, where *B* is the volume of data in each step. *B* takes the values of 256B, 64KB, and 1MB. Each process performs *N=500* steps. First, each process performs its **N** write steps, only performing puts and then with a new engine, so as to not share metadata, performs the same number of get operations. We exclusively time the I/O operations and we average the results across all processes on the system. At maximum scale, the application reads and writes 128GB. The Adios2 application runs over an OrangeFS instance on 16 nodes on NVMe and HDD.

Results are presented in Figure 8. On the put side, we can see that Hades' hierarchical management significantly outperforms single-layer I/O. This is achieved through the placement engine which is capable of maintaining free memory space for the application to write/read from data at almost all times, even when the dataset size is bigger than memory. The read side presents a more interesting picture, as we can see Hades suffers on the small Get operations. This is caused by the use of the OS cache by Adios2 which is not accessible to Hades as we self-manage all our data. This is rarely a problem, but for this application, the inherent overhead of Hades combined with the small I/O resulted in a comparatively higher impact on the I/O performance. For bigger Get operations, more common on HPC systems, we see the benefits that Hades brings.

3) *Metadata Management Performance*: Understanding the context and importance of data, and subsets of data, generated by an application is complex due to the performance of metadata management. Storing and extracting this information can be expensive and hinder the performance of the application. Hades proposes the use of a query system based on a SQL database to store, manage, relate, and query this metadata, which is crucial for helping analysis applications understand their data. This evaluation aims to show that, through careful use of asynchronous operations, an application can limit the impact of metadata management on the I/O path of the simulation while reaping significant rewards in the analytics phase.

For this evaluation, we compare the management of metadata through a file based metadata system and our SQL-based query system. To do so, each process executes *N* steps of a write phase, where three metadata inserts are executed per step: The App metadata, where one process, a node master, inserts the current step of the evaluation and the variables seen or calculated; the Variable metadata, where each process inserts the process-local metadata (shape, start, count, name, etc.) of any variable seen or calculated; the Hierarchical metadata, where each process inserts the current *Blob Location* for the given step, this value will later be modified by Fades hierarchical engine. Once finished, the processes execute the same number of steps on a read phase where all of them query the step, variable metadata, and Blob Location inserted on the same respective step. We set $N = 1k$ steps and we measure each operation with its own timer, aggregating across all processes at the application's end. The results are shown in Figure 9.

As expected, writing to a database, is significantly slower than performing I/O to a PFS. Yet, querying the system, through SQL queries, and not having to drill through the dataset in search of the desired information is almost infinitely more performant. These results motivate Hades' desire to make use of a database for analytic optimization and help to understand the asyn-

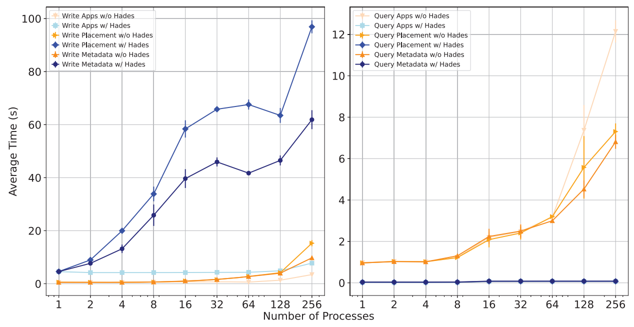


Fig. 9. Performance differences of varying metadata approaches

[1]anonymous_url

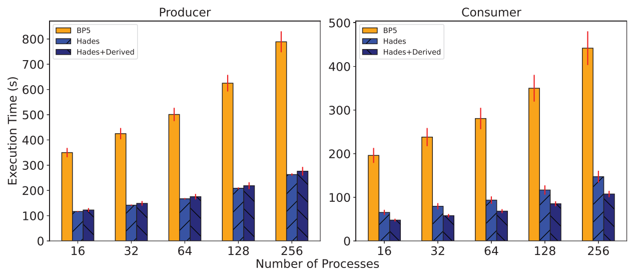


Fig. 10. Effects of calculating derived quantities on a producer/consumer workflow

chronous design of metadata management in the system, as our design removes the I/O overhead from the application I/O path.

*4) **The Benefit of Actively Deriving Quantities:*** To optimize for the analysis (consumer) phase, Hades actively computes derived quantities during the simulation (producer) phase. This can lead to performance advantages for the analysis phase, providing a reduction of total data movements and computing requirements. However, this can potentially penalize the simulation I/O path. This evaluation aims to quantify how actively deriving quantities impacts both the producer and consumer phases of a workflow.

For this evaluation, we run a synthetic application motivated by weather sciences. The producer phase creates a dataset representing a subdivided 3-D space. Each region of the space contains 3 variables: 3-D position, soil temperature, and outdoor temperature. The analysis phase aims to identify locations where the average soil temperature and outdoor temperature are larger than 25 degrees Celcius. If enabled, Hades actively calculates the derived quantity of ”average temperature”. We compare the execution time of this workload with Hades deriving quantities, without Hades deriving quantities, and without Hades. We run the experiment at varying scales to demonstrate the weak scalability of the derived quantity operators. Each application process generates 4GB of data in total, resulting in a 1TB total I/O at 256 processes.

Results are shown in figure 10. We can see that the effects of calculating the derived quantity are small but present. The slowdown in the simulation phase is mainly due to I/O contention and some CPU contention on the treads. Yet, the overall benefit of the analytic phase is significant, presenting a 20% speed up over the non-activated version as there is no need for extra calculation and reduced data movement on the aggregation step. Compared to the version without Hades, the application can achieve almost a 3x performance improvement on the analytic phase as it combines the benefits of both derived quantities, the hierarchy and the enriched metadata.

*5) **End-To-End Evaluation of Scientific Workflow:*** Scientific simulations oftentimes separate the production and analysis of data into distinct phases, requiring extensive data movements between the phases. In this evaluation, we aim to quantify the end-to-end performance improvement of using Hades to actively derive and hierarchically store quantities while data is generated. Additionally, we showcase Hades transparent design, as it has been built as an Adios2 engine, it can be used with any Adios-enabled workflow.

For this evaluation, we use the Gray-Scott [18] workflow,

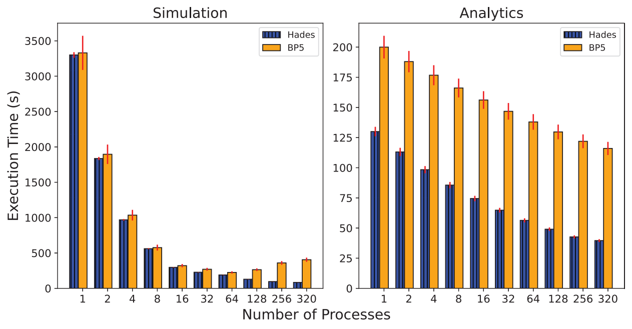


Fig. 11. Simulation and Analysis performance on the Gray-Scott workflow

which is a model of partial differential equations that captures the interactions between two chemical species reacting with each other and diffuse through a medium. It consists of two differential equations that describe how the concentration of these chemicals changes over time. The workflow produces a large dataset consisting of chemical concentrations, diffusion rates, and molecule positions within a 3-D cube of dimension L. This dataset is then read and analyzed to produce visualizations of the Probability Density Function(PDF), a statistical function that describes the likelihood of a continuous random variable taking on a particular value. When applied to the results of the Gray-Scott system, the PDF can be used to describe the distribution of concentration values over a given area or over the entire domain. We run a strong scaling experiment of the workflow. The size of the 3D stencil that defines the simulation (L) and the number of iterations (N) are increased to 256 and 1000, respectively. This leads to a global vector of doubles of size 256x256x256 with a total I/O of 125 GB. Workload checkpointing is disabled, but all other parameters are left on default. Runs not using Hades are executed with OrangeFS as a PFS on 16 nodes using NVMe.

The results are presented in Figure 11. We can see that both versions present, as expected, a very linear pattern over the strong scalability evaluation. We note that the Gray-Scott simulation is not data-bound limiting the benefits that Hades can bring to the execution time. On the other hand, the post-processing application, which calculates the PDF, requires extensive I/O consumption. Hades hierarchical optimization and the precalculation of the values for the computational phase resulted in a 3-4x speedup for this phase.

V. Discussions and Considerations

Balancing Computational Load: Hades effectively minimizes the time-to-insight by processing derived quantities during data production. However, this can increase computational demands, potentially straining the CPU, especially when derived calculations are complex. While the current implementation takes this into account, it does so for simpler operators. Thus, there is a need to explore the limitations and trade-offs when using operations that Hades does not currently cover. As an open-source platform, Hades offers the scientific community an opportunity to investigate these aspects further.

Optimizing Computation Placement: While many workflows read significant more data than they write [25], [26], this is not globally the case. Under this circumstances, Hades derived quantity calculation might not be optimal. Yet, our operational metadata can be a solution for these scenarios as it enables the identification and location of specific data segments for on-demand computation of derived quantities. Exploring this tradeoff – the choice between in-transit computation versus on-demand computing when data is requested – opens up possibilities for enhancing Hades' adaptability and performance in diverse and complex workflow structures.

Adapting to Diverse I/O Libraries: Currently, Hades' design and architecture is validated using the Adios engine infrastructure, which may be limiting in operational functionality. Scientific computing uses a wide plethora of high-level I/O libraries, data formats, and data models. To allow future extensibility, Hades offers a community-driven plugin layer where few basic APIs are offered to developers for extending their libraries to use Hades. For example, libraries such as pNetCDF and HDF5 use similar self-describing capabilities [27], [28] and have separated their APIs from their storage engines, similar to ADIOS [29]. Hence, one would simply inherit our `read()-write()-define()-query()` functions through the API interception to bring Hades into other libraries.

VI. Related Work

Indexing Techniques in Databases: Indexing is a fundamental aspect of databases, ensuring efficient data access. Two prevalent indexing techniques are R-tree [30] and bitmaps [31]. These techniques are not just confined to traditional databases but have also found applications in software products tailored for scientific data, such as SciDB [32] and FastBit [33]. These products leverage these indexing techniques to expedite queries over extensive scientific datasets. However, these techniques is the overhead they introduce, especially when dealing with high-velocity, voluminous scientific data.

I/O Libraries in Scientific Applications: Data-intensive scientific applications often resort to specialized I/O libraries for data storage. Libraries like HDF5 [3], ADIOS [4], [5], and NetCDF [6] are popular choices. These libraries not only store data but also associate it with metadata, enhancing future data access efficiency. Yet, a lingering challenge remains: these libraries do not adequately cater to the scientists' requirement to query through every element of their datasets. Systems such as DataStager [34] which leverage asynchronous movement of data strategically timed to occur when the application is engaged in computational tasks, thereby minimizing I/O wait times and enhancing overall system efficiency. Similarly, PnetCDF Staging [35] provides similar optimizations. but focused on collective parallel I/O for PnetCDF users.This systems do not incorporate active storage components or offer management capabilities for derived quantities.

First-Generation Metadata Tools: One of the pioneering tools in this category is Starfish [36]. These tools primarily concentrated on the storage of file-level metadata. Their main function was to capture and store descriptive information about data files, without delving deeper into the actual content or structure of the data itself.

Second-Generation Metadata Tools: Advancements in metadata management led to the development of second-generation tools, exemplified by systems like SciDB [32] and FastBit [33]. These tools went a step further by offering direct indexing of data files. Such a feature is instrumental in facilitating the discovery of raw data values within large datasets. Instead of just describing the data, these tools provided mechanisms to quickly locate specific data values within files.

Third-Generation Metadata Tools: The third generation witnessed a more sophisticated approach to metadata management. Tools such as SoMeta [37], TagIt [38], BIMM [39], SPOT Suite [40], JAMO [41], MIQS [42], and Empress [43] introduced feature indexing in a Key-Value Store (KVS) style. Notably, MIQS adopted a unique approach by extending the data file itself, rather than relying on an external database. While this method simplifies the architecture, it might not offer the advanced functionalities typically associated with dedicated database systems.

VII. Conclusions and Future Work

In scientific workflows, distinct production and analysis phases often lead to substantial data movements, hindering timely insight generation. Hades addresses this challenge by optimizing input/output processes through intelligent management of deep memory and storage hierarchies. Comparative evaluations demonstrate Hades' superior performance over conventional systems in large-scale data operations, attributed to its hierarchical management approach. Its novel metadata management strategy leveraging enriched metadata and a querable API, enhances analytics performance, albeit with a minimal increase in data input overhead. Significantly, Hades computes derived quantities concurrently with data production, offering considerable reduction in complexity and performance increase in the analysis phase. This efficiency is validated through tests on both synthetic and real-world applications, including the Gray-Scott workflow, with Hades achieving performance improvements of 3-4x in real-world scenarios. These results establish Hades as a groundbreaking tool in scientific computing.

As part of our future work, we aim to expand Hades by introducing new operators for derived quantities. This expansion will encompass complex mathematical operations such as partial differential equations, eigenvalues, and integrals, broadening its applicability across various scientific domains. Moreover, we plan to optimize HDCalc's performance by enabling the execution of certain operators on computation accelerators, including GPUs and FPGAs. This shift will reduce the CPU load and facilitate the implementation of more sophisticated and efficient operators. A significant enhancement will also involve the integration of SQL support into Hades. Given SQL's extensive use in data query and analysis, developing a parser that transforms SQL queries into OpGraphs will significantly improve Hades' usability. This integration aims to attract a wider user base proficient in SQL, thereby aligning Hades with modern data processing and analysis requirements.

Acknowledgment

This work is supported by the U.S. Department of Energy (DOE) under DE-SC0023263.

References

[1] J. Cernuda, H. Devarajan, L. Logan, K. Bateman, N. Rajesh, J. Ye, A. Kougkas, and X.-H. Sun, "Hflow: A dynamic and elastic multi-layered i/o forwarder," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 114–124.

[2] P. Harrington, W. Yoo, A. Sim, and K. Wu, “Diagnosing parallel i/o bottlenecks in hpc applications,” in *International Conference for High Performance Computing Networking Storage and Analysis (SCI7) ACM Student Research Competition (SRC)*, 2017, p. 4.

[3] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, “An overview of the hdf5 technology suite and its applications,” in *Proceedings of the EDBT/ICDT 2011 workshop on array databases*, 2011, pp. 36–47.

[4] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, A. Huebl, M. Kim, J. Kress, T. Kurc, Q. Liu, J. Logan, K. Mehta, G. Ostrouchov, M. Parashar, F. Poeschel, D. Pugmire, E. Suchyta, K. Takahashi, N. Thompson, S. Tsutsumi, L. Wan, M. Wolf, K. Wu, and S. Klasky, “ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management,” *SoftwareX*, vol. 12, p. 100561, 2020.

[5] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, “Flexible io and integration for scientific codes through the adaptable io system (adios),” in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, 2008, pp. 15–24.

[6] R. Rew and G. Davis, “Netcdf: an interface for scientific data access,” *IEEE computer graphics and applications*, vol. 10, no. 4, pp. 76–82, 1990.

[7] L. Logan, J. C. Garcia, J. Lofstead, X. Sun, and A. Kougkas, “Labstor: A modular and extensible platform for developing high-performance, customized i/o stacks in userspace,” in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15.

[8] C.-G. Lee, H. Byun, S. Noh, H. Kang, and Y. Kim, “Write optimization of log-structured flash file system for parallel i/o on manycore servers,” in *Proceedings of the 12th ACM International Conference on Systems and Storage*, 2019, pp. 21–32.

[9] D. M. Dunlavy, D. P. O’Leary, J. M. Conroy, and J. D. Schlesinger, “Qcs: A system for querying, clustering and summarizing documents,” *Information processing & management*, vol. 43, no. 6, pp. 1588–1605, 2007.

[10] S. Yang, Y. Xie, Y. Wu, T. Wu, H. Sun, J. Wu, and X. Yan, “Slq: A user-friendly graph querying system,” in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 893–896.

[11] U. Ayachit, A. Bauer, B. Geveci, P. O’Leary, K. Moreland, N. Fabian, and J. Mauldin, “Paraview catalyst: Enabling in situ data analysis and visualization,” in *Proceedings of the first workshop on in situ infrastructures for enabling extreme-scale analysis and visualization*, 2015, pp. 25–29.

[12] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, K. Bonnell, M. Miller, G. Weber, C. Harrison, D. Pugmire, T. Fogal, C. Garth, A. Sanderson, E. W. Bethel, M. Durant, D. Camp, J. Favre, O. Rübel, P. Navratil, and F. Vivodtzev, “Visit: An end-user tool for visualizing and analyzing very large data,” *Proceed SciDAC*, pp. 1–16, 01 2011.

[13] J. Gu, S. Klasky, N. Podhorszki, J. Qiang, and K. Wu, “Querying large scientific data sets with adaptable io system adios,” in *Supercomputing Frontiers: 4th Asian Conference, SCFA 2018, Singapore, March 26-29, 2018, Proceedings 4*. Springer International Publishing, 2018, pp. 51–69.

[14] Z. Lan, Z. Zheng, and Y. Li, “Toward automated anomaly identification in large-scale systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 174–187, 2009.

[15] A. Kougkas, H. Devarajan, and X.-H. Sun, “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*, 2018, pp. 219–230.

[16] T. Wang, S. Byna, B. Dong, and H. Tang, “UniviStor: Integrated Hierarchical and Distributed Storage for HPC,” in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. USA: IEEE, 2018.

[17] B. Dong, S. Byna, K. Wu, H. Johansen *et al.*, “Data elevator: Low-contention data movement in hierarchical storage system,” in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*. Hyderabad, India: IEEE, 2016.

[18] J. E. Pearson, “Complex patterns in a simple system,” *Science*, vol. 261, no. 5118, pp. 189–192, 1993. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.261.5118.189

[19] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, Z. Liu, J. Berner, W. Wang, J. G. Powers, M. G. Duda, D. M. Barker, and X.-Y. Huang, “A description of the advanced research wrf model version 4,” UCAR/NCAR, Tech. Rep., 2019. [Online]. Available: https://opensky.ucar.edu/islandora/object/opensky:2898

[20] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh *et al.*, “Sympy: symbolic computing in python,” *PeerJ Computer Science*, vol. 3, p. e103, 2017.

[21] I. Berg, “muparser-a fast math parser library,” 2011.

[22] A. Shoshani and D. Rotem, “Scientific data management. challenges, technology, and development,” *Scientific Data Management: Challenges, Technology, and Deployment*, 12 2009.

[23] W. Shi, P. Cheng, C. Zhu, and Z. Chen, “An intelligent data placement strategy for hierarchical storage systems,” in *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, 2020, pp. 2023–2027.

[24] Gray-scott simulation code. [Online]. Available: https://github.com/pnorbert/adiosvm/tree/master/Tutorial/gray-scott

[25] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, “Characterization of scientific workflows,” in *2008 Third Workshop on Workflows in Support of Large-Scale Science*, 2008, pp. 1–10.

[26] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, “Characterizing and profiling scientific workflows,” *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013, special Section: Recent Developments in High Performance Computing and Security. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X12001732

[27] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, “An overview of the hdf5 technology suite and its applications,” in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, ser. AD ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 36–47. [Online]. Available: https://doi.org/10.1145/1966895.1966900

[28] C. S. Zender, “Analysis of self-describing gridded geoscience data with netcdf operators (nco),” *Environmental Modelling & Software*, vol. 23, no. 10, pp. 1338–1342, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364815208000431

[29] J. Soumagne, J. Henderson, M. Chaarawi, N. Fortner, S. Breitenfeld, S. Lu, D. Robinson, E. Pourmal, and J. Lombardi, “Accelerating hdf5 i/o for exascale using daos,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 903–914, 2022.

[30] Y. Theodoridis and T. Sellis, “A model for the prediction of r-tree performance,” in *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1996, pp. 161–171.

[31] C.-Y. Chan and Y. E. Ioannidis, “Bitmap index design and evaluation,” in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998, pp. 355–366.

[32] M. Stonebraker, P. Brown, D. Zhang, and J. Becla, “Scidb: A database management system for applications with complex analytics,” *Computing in Science & Engineering*, vol. 15, no. 3, pp. 54–62, 2013.

[33] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann *et al.*, “Fastbit: interactively searching massive data,” in *Journal of Physics: Conference Series*, vol. 180, no. 1. IOP Publishing, 2009, p. 012053.

[34] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, “Datastager: scalable data staging services for petascale applications,” *Cluster Computing*, vol. 13, no. 3, pp. 277–290, 2010.

[35] J. Lofstead, R. Oldfield, T. Kordenbrock, and C. Reiss, “Extending scalability of collective io through nessie and staging,” in *Proceedings of the sixth workshop on Parallel Data Storage*, 2011, pp. 7–12.

[36] Starfish Storage, “Starfish Storage — Metadata & Rules Framework for Large-Scale File Management,” https://starfishstorage.com, June 2022.

[37] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, “Someta: Scalable object-centric metadata management for high performance computing,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 359–369.

[38] H. Sim, Y. Kim, S. S. Vazhkudai, G. R. Vallée, S.-H. Lim, and A. R. Butt, “Tagit: an integrated indexing and search service for file systems,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[39] D. Korenblum, D. Rubin, S. Napel, C. Rodriguez, and C. Beaulieu, “Managing Biomedical Image Metadata for Search and Retrieval of Similar Images,” *Journal of digital imaging*, vol. 24, no. 4, pp. 739–748, 2011.

[40] T. Craig E., E. Abdelilah, G. Dan *et al.*, “The SPOT Suite Project,” http://spot.nersc.gov/.

[41] Joint Genome Institute, “The JGI Archive and Metadata Organizer(JAMO),” http://cs.lbl.gov/news-media/news/2013/new-metadata-organizer-streamlines-jgi-data-management.

[42] W. Zhang, S. Byna, H. Tang, B. Williams, and Y. Chen, “Miqs: Metadata indexing and querying service for self-describing file formats,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356146

[43] M. Lawson and J. Lofstead, “Using a robust metadata management system to accelerate scientific discovery at extreme scales,” in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018, pp. 13–23.