WisIO: Automated I/O Bottleneck Detection with Multi-Perspective Views for HPC Workflows

Izzet Yildirim
Illinois Institute of Technology
Chicago, IL, USA
iyildirim@hawk.iit.edu

Hariharan Devarajan
Lawrence Livermore National Laboratory
Livermore, CA, USA
hariharandev1@llnl.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

Kathryn Mohror
Lawrence Livermore National Laboratory
Livermore, CA, USA
kathryn@llnl.gov

Abstract

Modern HPC workloads involve large data transfers that can become bottlenecks. Existing analysis tools identify bottlenecks from per-file performance data but have limitations in parallelizability and rigid heuristic-based rules, necessitating an automated, efficient, and multi-perspective solution. We designed an automated tool, WisIO, that enables parallel and distributed analysis of multi-terabyte-scale workflow performance data. WisIO examines performance data from multiple perspectives, uses metric-driven bottleneck classification, and allows extensible mapping of bottlenecks to root causes. Experimental results demonstrate that WisIO's multi-perspective views substantially improve bottleneck coverage, showing an average increase of up to 805× when compared to analyzing performance data from a single perspective. In our performance evaluation, WisIO's metric-driven classification processed 340K bottlenecks per second, while its reasoning engine handled around 35K bottlenecks per second. In an analysis of five real-world HPC workloads, WisIO demonstrated up to 11× faster analysis time and identified up to 144× more bottlenecks compared to existing solutions.

CCS Concepts

• **Information systems** → **Information storage systems**; • **General and reference** → *Performance*; • **Software and its engineering** → Software libraries and repositories.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725742

Keywords

HPC, Workflows, I/O Analysis, I/O Bottleneck Detection

ACM Reference Format:
Izzet Yildirim, Hariharan Devarajan, Anthony Kougkas, Xian-He Sun, and Kathryn Mohror. 2025. WisIO: Automated I/O Bottleneck Detection with Multi-Perspective Views for HPC Workflows. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721145.3725742

1 Introduction

Transferring large amounts of data is a common task within modern scientific high-performance computing (HPC) workloads such as simulations, artificial intelligence (AI) applications, and workflows involving multiple cooperating applications [8, 20, 41]. Some examples of data transfers include reading input files from storage systems at the start of an application [38], transferring data between producers and consumers, and checkpointing critical workload data for fault tolerance. These data transfers are crucial for enabling scientific discoveries across many HPC workloads. However, the large volume of these data transfers often lead to workloads spending significant time performing I/O [14, 16, 25, 29]. As a result, tuning the performance of data transfers has become a routine task for application developers [19, 31, 36, 46]. This tuning involves analysis of I/O performance data to extract critical paths for data transfer within the scientific workloads [9].

State-of-the-art tools for I/O analysis utilize I/O performance data from I/O profilers and tracers such as Darshan [33] and Recorder [43]. Popular I/O analysis tools that work on single application performance data include UMAMI [28], TOKIO [27], IOMiner [45], and Drishti [3]. These tools utilize a serial pipeline using a Pandas-like interface to perform analysis on per-file I/O

performance data [3, 28, 31, 44]. In a recent study on workflows, Devarajan et al. performed a manual I/O bottleneck analysis to study different aspects of the workflow, such as timeline (one-second intervals), processes, and files, to visualize I/O behaviors [9]. The study demonstrated that different aspects of the workflow can highlight different I/O bottlenecks. Additionally, these tools employ rule-based mechanisms, such as mathematical models [28, 45, 49, 50], metrics [27, 28, 45, 49, 50], and heuristics [3, 9], to identify certain I/O behaviors as bottlenecks. In general, these solutions have demonstrated their capability to identify and optimize I/O for individual workloads on HPC systems.

While state-of-the-art tools assist in I/O bottleneck detection, they have several limitations. First, *identifying bottlenecks through serial analysis of performance data from large-scale workflows is extremely time-consuming.* For instance, a serial pipeline analyzing a 400 GB trace data is 7× slower than a parallel pipeline [48]. Second, *identifying bottlenecks in an isolated part of the workflow (e.g., performance data from a single process) can miss performance problems in other areas.* For instance, analyzing CM1 (an atmospheric-simulation workload) performance data using per-process and per-file behaviors reveals different bottlenecks [9]. Third, *identifying bottlenecks using behavior-driven heuristics might miss unseen behavior in new workloads.* For instance, opening multiple files from eight nodes simultaneously can lead to an I/O bottleneck in GPFS [40] but not in Lustre [23], making behavioral heuristics unreliable for bottleneck detection. Finally, *existing solutions attempt to explain a bottleneck using a single heuristic, overlooking additional underlying factors.* For instance, within 1000 Genomes (a data-intensive workflow to uncover potential disease-related mutations), the mutation overlap application has both metadata and small write issues on the same file. Therefore, we need a fully automated bottleneck detection tool to analyze multiple perspectives, perform behavior-independent classifications, and perform root-cause analysis for large-scale HPC workflows.

We introduce WisIO (Wisdom from I/O Behavior), an automated bottleneck detection tool for large-scale HPC workflows. WisIO creates a composable task graph for identifying bottlenecks that can be efficiently pipelined and parallelized over distributed resources. WisIO analyses the trace data from large-scale workflows by partitioning it into multiple chunks and operating on it in parallel across multiple workers. This process uses the Dask parallel computing library to manage the trace data in distributed memory and apply analysis using composable tasks. As a part of the composable analysis tasks, WisIO has three main stages. In the first stage, WisIO utilizes *multi-perspective views* to create multiple viewpoints, such as per-file, per-process, and timeline, of the workflow's trace data to extract different workload behaviors for bottleneck discovery. In the second stage, WisIO identifies the bottlenecks using a classification algorithm that replaces behavior-specific heuristics with metric-driven and relativistic severity calculation to classify bottlenecks. Finally in the last stage, WisIO uses an extensible rule engine that performs root cause analysis on the classified bottlenecks by attaching one or more observed behaviors as reasons. The evaluation results demonstrate that WisIO is parallelizable across eight nodes and detects 144× more bottlenecks in 11× smaller analysis time for the Montage workflow as compared to state-of-the-art analysis solutions. The main contributions of this work are:

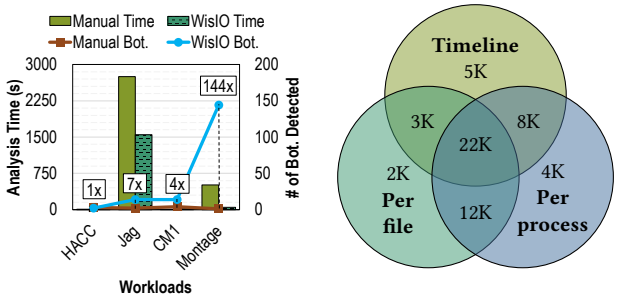
- *Design of WisIO* for efficiently analyzing multi-terabyte-scale workflow performance data over distributed resources
- *Design of multi-perspective views* for building analysis-driven perspectives of the same performance data to increase the bottleneck classification coverage;
- *Design of a metric-driven bottleneck classification algorithm* that decouples bottleneck detection from behavioral heuristics to classify bottlenecks for unseen performance data;
- *Design of a reasoning engine* that performs root cause analysis on classified bottlenecks;
- *Evaluation of WisIO with large-scale workflows* demonstrates WisIO can identify 144× more bottlenecks and in 11× lower time as compared to state-of-the-art solutions.

2 Background & Related Work

Tuning I/O performance involves analysis of performance data collected from scientific workloads. This analysis is performed by HPC tools that can be categorized into single-application and system-wide I/O analysis tools. The single-applications tools such UMAMI [28], IOMiner [45], Drishti [3], and AIIO [15] extract behavioral heuristics and detect bottlenecks using workload's performance data. In contrast, TOKIO [27] is a system-wide tool that extracts system-level behavioral heuristics and detects bottlenecks. These analysis tools use performance data extracted by collection tools such as I/O profilers [7, 22] that collect aggregated performance counters and I/O tracers [11, 42, 47] that collect event-level information. These tools facilitate the understanding and tuning of scientific HPC workloads.

Using an in-memory analysis pipeline, existing analysis tools detect I/O bottlenecks from a workload's aggregated performance data. In contrast, WisIO advocates using detailed I/O traces with a loosely coupled staged analysis pipeline that can efficiently work with large workflow traces. Three main stages of WisIO which differentiate it from all existing analysis tools are the creation of multi-perspective views, metric-driven classification algorithm, and rule-based reasoning engine.

Current bottleneck detection solutions use performance data from I/O profilers to utilize per-file performance



(a) Manual analysis vs. WisIO. (b) Perspective bottlenecks.

Figure 1: (a) WisIO detects 144× more bottlenecks in 11× less time as compared to baseline. (b) Different bottlenecks can be uncovered from different perspectives.

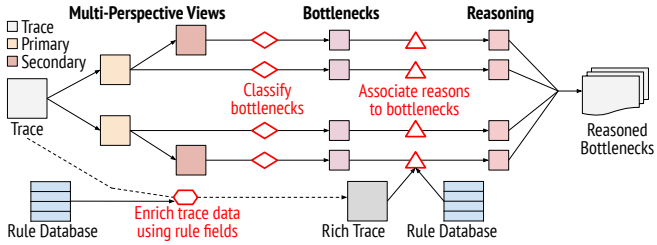


Figure 2: High-level execution flow of WisIO, showcasing individual stages and their parallel and pipelined execution flow.

counters. For example, tools use Darshan counters like POSIX_SIZE_READ_100K_1M to detect bottlenecks caused by small read operations. However, in this work, the multi-perspective views demonstrate that bottlenecks caused by small reads may not always be evident from the file-level data but may require application-level data in certain cases. Other solutions, such as [4, 9, 30], use the trace data to manually create per-process and timeline aggregations to identify unique behaviors. However, WisIO automates the multi-perspective views, allowing users to dynamically specify these views as well as combines these views to create complex views for bottleneck detection.

Existing solutions detect bottlenecks by coupling both the behavior and the metric extracted for the workload. This coupling reduces the bottleneck search space to a smaller set of candidates based on historical knowledge. Subsequently, the metrics are used to explain newly discovered behaviors. In contrast, WisIO's bottleneck classification algorithm uses the metric to detect potentially problematic accesses without considering the behavioral aspect of the workload.

Finally, existing solutions utilize a one-to-one mapping to link each bottleneck with a specific explanation. For instance, Drishti identifies small I/O accesses based on a predefined threshold for access sizes (e.g., smaller than 100 KB or 1 MB) [3]. In contrast, WisIO's rule-based reasoning engine operates on the identified bottlenecks and performs root cause analysis to find multiple potential reasons.

3 Motivation

State-of-the-art bottleneck detection solutions present three challenges that motivated us to develop WisIO. First, these solutions use a serial pipeline for bottleneck detection that would be impractical with terabyte-scale traces [9]. To illustrate the problem, we replicate the manual I/O analysis performed by Devarajan et al. [9], where five bottleneck detection queries are executed serially against four real HPC workloads (for more details see Table 2). We observe that the serial pipeline is 11× slower and detects 144× less critical bottlenecks compared to WisIO (Figure 1a). Second, current tools rely on the singular per-file performance data provided by I/O profilers [33, 47]. However, our initial analysis of the Montage workflow's performance data indicates that focusing solely on a per-file perspective may not provide a complete understanding of the workflow's I/O issues. We observe that 2K bottlenecks are detected using the per-file perspective, whereas 5K bottlenecks are detected using the timeline perspective (Figure 1b). Third, current solutions map each bottleneck to a single reason which might miss other potential root causes. For instance, WisIO detects three different reasons (i.e., excessive metadata access, operation imbalance, and size imbalance) for a critical bottleneck during the 7th second of HACC's data exchanges (as shown in CR1 of Figure 17).

4 WisIO

WisIO is an automated I/O bottleneck detection tool to pinpoint I/O behaviors that cause bottlenecks and heuristically identify their root causes in large-scale HPC workflows. Operating offline, WisIO analyzes trace data captured by I/O tracers (e.g., Recorder [42], DFTracer [11]), enabling bottleneck detection without adding overhead to workloads. It utilizes three main stages, namely multi-perspective views, a metric-driven bottleneck classification algorithm, and an extensible rule engine to detect bottlenecks within workflows (Figure 2).

First, constructing multi-perspective views depending on the user-specified perspectives allows users to look at the same trace data from different angles (Section 4.1). For instance, a function name perspective allows WisIO to create a view for examining behaviors across different I/O functions. In contrast, an interface name perspective allows WisIO to generate a view for examining different behaviors across interfaces. By default, WisIO uses the process name, file name, and timeline perspectives. Second,

identifying behaviors within perspectives using a classification algorithm that replaces behavior-specific heuristics with metric-driven bottleneck selection (Section 4.2). This allows WisIO to classify previously unseen bottlenecks, enabling the discovery of new patterns that emerge in evolving HPC workloads. Third, associating heuristic-driven reasons for the classified bottlenecks using an extensible rule engine that can map one or more behavioral reasons to bottlenecks (Section 4.3). These behavioral reasons are extracted from an enriched trace data to accommodate rules and reasoning conditions within WisIO. All these stages are designed as a composable task graph for efficient pipelining and parallelization across distributed resources. The output of these stages is the reasoned bottlenecks that are available for further investigation and optimization efforts.

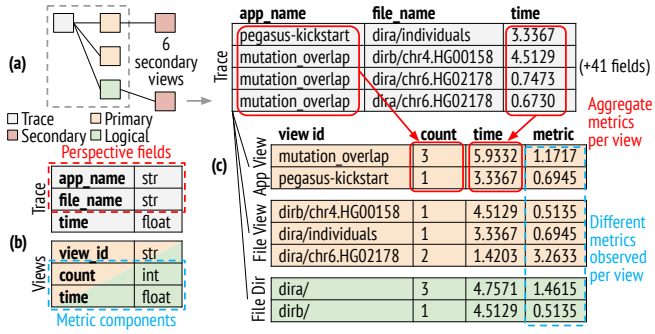


Figure 3: (a) Focused primary and logical views. (b) Data types of perspective fields and metric components. (c) Different metrics are observed per view.

4.1 Multi-Perspective Views

I/O profilers record I/O performance data on a per-file basis [33, 47]. Current bottleneck detection solutions identify bottlenecks using this per-file performance data. However, analyzing only per-file performance data may not provide a complete understanding of I/O behavior (see Figure 1b in Section 3). To overcome this limitation, we introduce multi-perspective views that increases the bottleneck classification coverage by utilizing the detailed trace data from I/O tracers to dynamically construct user-specified perspectives from the same trace data, instead of relying solely on per-file performance data. However, trace data contains millions of I/O operations and is not aggregated like the performance data from I/O profilers. Therefore, aggregating the trace data into view-specific performance data is essential for retaining only relevant I/O behavior and metrics for each perspective.

Multi-perspective views within WisIO are of three types: primary, logical, and secondary views. The primary view is the foundational perspective directly constructed from the trace data to spotlight specific I/O behavior relevant to a chosen perspective. The logical views are constructed from the derived columns of the trace data that represent a new characteristic of the I/O behavior. For instance, a file directory logical view, derived from the file name column, would show specific I/O behavior to file directories as compared to individual files. These views allow users to choose domain-driven perspectives for bottleneck analysis.

Finally, the secondary view is a combination of multiple primary and logical views in a specified order to identify I/O behavior specific to the relationship between these primary views (e.g., combining process name and file name perspectives to see how specific processes interact with files). In total, a total of $\sum_{r=1}^{n} \frac{n!}{(n-r)!}$ secondary views are constructed for a given set of n primary and logical views. For instance, the initial set of process name and file name perspectives (i.e., $n = 2$) would yield a total of four views. All these views contain multiple records with view id as their identifier for aggregation and metric as their value. Some examples of metrics are I/O time, I/O operations per second (IOPS), and I/O bandwidth (BW). The I/O time metric can be maintained and calculated easily during aggregation. In contrast, the IOPS and BW metrics need to be calculated by aggregating their components correctly. For instance, we need to aggregate I/O time and number of operations for every perspective record to compute IOPS. Therefore, WisIO's view contains the identifier view id and the metric components as values for each perspective record in a Dask DataFrame.

To demonstrate how different I/O behaviors emerge through multi-perspective view analysis, we present an example in Figure 3. As shown in the figure, the trace data consists of 44 fields, with the app_name, file_name, and time fields explicitly displayed, while other 41 fields, such as the I/O category (read, write, and metadata), are omitted for brevity (Figure 3c). We begin by aggregating the trace data into two primary views (application name and file name perspectives) and one logical view (file directory perspective) using the app_name, file_name, and file_dir fields, respectively. For this example, we use IOPS as our metric, calculated from I/O time and I/O operations (shown as time and count fields in the figure). We then combine these views to form six secondary views based on our formula described above. Throughout all views, we maintain the metric components for IOPS, namely I/O time and number of operations. As demonstrated in the figure, the aggregation results differ across views. For instance, while directory dira has the highest IOPS in the file directory view, the file name view reveals that only one file in that directory (dira/chr6.HG02178) actually contributes to the high IOPS. This demonstrates how analyzing trace data from multiple perspectives can uncover distinct workload characteristics that might be hidden in a single view.

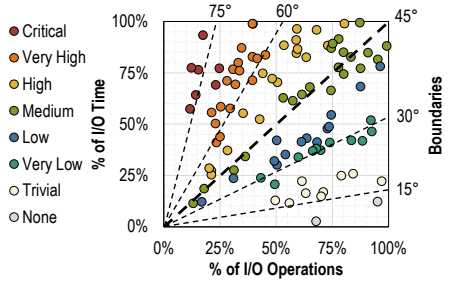


Figure 4: In the case of the I/O operations per second (IOPS) metric, lower metrics indicate higher severity. Notably, $> 75^\circ$ indicates critical bottlenecks.

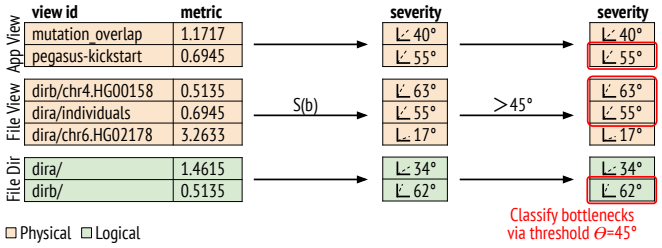


Figure 5: Different behaviors are classified as bottlenecks depending on the perspective. e.g., the App View indicates the *pegasus-kickstart* application exhibits a bottleneck, while the *mutation_overlap* does not.

4.2 Metric-Driven Bottleneck Classification

Existing bottleneck detection solutions utilize behavioral heuristics to identify bottlenecks within the trace data (see Section 2). These heuristics result in a tight coupling between the workload behaviors and the identified bottlenecks. However, this tight coupling cannot detect previously unseen bottlenecks from the growing diversification of workloads [9, 12] and system behaviors. For instance, random accesses on disk-based parallel file system is a bigger problem than on flash-based node-local burst buffers. Another instance, I/O bottlenecks on simulations with bulk synchronous operations are behaviorally different from independent I/O on AI workloads [12] and therefore may or may not be a bottleneck on the HPC system. Additionally, manually creating heuristics for all possible bottlenecks for existing and future workloads is an insurmountable task.

In the literature, metrics such as I/O bandwidth and I/O time have been widely used in analysis to manually detect I/O bottlenecks [1, 21, 28, 45, 46, 49, 50]. These metrics do not depend on behavioral aspects on the workload and can be used independently of the workload characteristics to detect bottleneck. However, doing this manually, especially for the combinatorial set of perspectives, is extremely expensive. This dictates the need for an automated metric-driven bottleneck identification algorithm that is decoupled from workload's behavioral heuristics and can be efficiently applied to multiple perspectives.

We introduce a novel classification algorithm that leverages the severity parameter to identify "how severe a metric is for a given perspective record?". The severity parameter within the classification algorithm has two properties: monotonically increasing and relativistic. The first property allows the algorithm to classify the perspective records independently. The second property compares the relative performance of perspective records. In the context of I/O bottleneck classification, we use I/O metrics such as I/O time, IOPS, and I/O bandwidth to define severity for the algorithm. As these metrics may satisfy the properties, we apply transformations on these metrics to ensure they satisfy the severity parameter. In this work, we choose IOPS as our metric. To quantify the severity of each perspective record using IOPS, we employ the formula $S(b) = \tan \frac{\%\text{ of I/O time}}{\%\text{ of I/O operations}}$. Here, IOPS metric has been inverted to fit the definition of the severity parameter. Additionally, we calculate the slope of the perspective record (i.e., tan) to intuitively bound the severity parameter between 0° and 90°. In this formula, a smaller value of the severity parameter signifies a smaller bottleneck and vice-versa. After calculating the severity $S(b)$, the algorithm proceeds to classify bottlenecks by comparing the calculated severities against a threshold degree. The threshold value for the severity parameter depends on the underlying metric. For the IOPS metric, we set the default threshold angle θ at 45° ($\frac{\pi}{4}$ or $tan(\theta) = 1$). This choice of 45 degrees represents the point where the percentage of I/O operations attributed to a record equals the percentage of total I/O time, with higher degrees indicating increased severity of the bottleneck due to the higher proportion of I/O operations relative to the record's I/O time. By applying this threshold, the algorithm classifies certain perspective records as bottlenecks ($S(b) > \theta$) or non-bottlenecks ($S(b) <= \theta$).

We demonstrate the classification algorithm's output for a microbenchmark to illustrate an example of this threshold for IOPS (Figure 4). In the example, the perspective records are classified based on the value of the severity parameter. Here, a smaller severity parameter signifies a higher IOPS value (white to green) and vice versa (red to orange). Therefore, for a threshold greater than 45°, we will classify all perspective records as bottlenecks where the I/O time is greater than the number of operations. The boundaries of the severity parameter are mapped to user-facing labels that improve the presentation of bottlenecks for WisIO. In the context of IOPS, the "critical" label is mapped to $> 75^\circ$, the "very high" label is $> 60^\circ$, the "high" label is $> 45^\circ$, and the rest are labeled as medium and low bottlenecks.

To illustrate a working example of the algorithm, we continue our example from the previous section. In the previous

section, we had calculated the IOPS metric for the perspective records. Now, we are applying our classification algorithm on the three views: two primary and one logical (Figure 5). For calculating the severity parameter, we invert the IOPS metric and then calculate the slope of the line using the tangent function. This gives us a severity parameter between 0° and 90°. In the figure, the File View has three files with different severity parameters. The severity parameters for the files dirb/chr4.HG00158, dira/individuals, and dira/chr6.HG02178 are 63°, 55°, and 17°. This indicates that dirb/chr4.HG00158 has a bigger bottleneck than the other two files when considering the IOPS metric. With this applied to all the perspective records, we see that out of seven perspective records, four of them are classified as bottlenecks by the algorithm. This illustration shows that the perspective records can be classified independently and thus be parallelized efficiently across multiple analysis workers for large-scale trace data.

4.3 Rule-Based Bottleneck Reasoning

After the bottlenecks are classified, WisIO aims to analyze the root cause of the bottlenecks by mapping the bottlenecks with the workload's I/O behavior. This mapping is performed by using a rule engine that utilizes workload behaviors to reason about the identified bottlenecks. This rule engine is backed by a database of heuristic rules with workload behaviors as variables to find reasons for the classified bottlenecks. A rule is defined as a typed expression language with a custom grammar for creating boolean conditions that match a reason based on workload behavior. Each rule is defined by a key, a name, and a condition. The key is used to identify, and if needed override, the rule. The name is used for presentation. The condition is written as a boolean expression using the field names in the trace data. A sample rule definition for the small imbalance rule is presented in Listing 1.

```
size_imbalance:
  name: "Size imbalance"
  condition: "(abs(write_size - read_size) / size) > 0.1"
  reasons:
    - condition: "read_size > write_size"
      message: "'read' size is {{read_size}} which ..."
```

Listing 1: Sample definition of the size imbalance rule.

Furthermore, each rule has their own reasoning, defined by a condition and a message. For example in Listing 1, the *size imbalance* rule examines whether read or write size exceeds the other by 10%. If this condition is met, then the rule investigates the reasoning conditions to identify the specific reason behind the bottleneck. These rules are created from a literature review of analysis tools that provide a collection of behaviors known to be bottlenecks for specific HPC systems [3, 9, 26, 44] and are presented in Table 1.

Table 1: Rules and conditions implemented in WisIO.

Rule	Condition
Small reads	Over 50% of I/O time is spent on small (<1 MiB) read operations
Small writes	Over 50% of I/O time is spent on small (<1 MiB) write operations
Excessive metadata access	Over 50% of I/O time is spent on metadata operations
Operation imbalance	Read or write operations exceeds the other by 10%
Size imbalance	Read or write size exceeds the other by 10%

Attaching rules to classified bottlenecks involves the matching of rules, which is a cross-product of the number of rules, the number of reasons per rule, and the classified bottlenecks. This combinatorial space of matching results in an expensive rule engine that requires a lot of resources to compute this cross-product. Additionally, the rule engine needs to access the enriched trace data to apply these rules on different perspective records. To optimize this process, the rule engine performs these matches in parallel using the Dask task graph and the enriched trace data is persisted in distributed memory using Dask DataFrames. Essentially, the rule engine can execute every combination of the cross-product independently of each other and access the enriched data concurrently from these independent tasks.

Enriched data plays a critical role in attaching rules to classified bottlenecks. It is derived from the original trace data with the goal of capturing workload behaviors relevant to the perspectives selected by the user. Since the rules rely on workload behaviors as variables, the enriched data must encompass all the necessary behaviors for the rule engine to effectively apply its rules and reasoning.

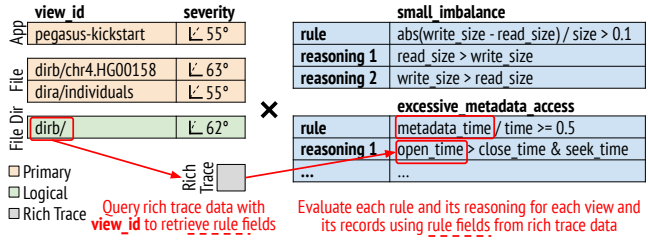


Figure 6: Full diagnosis of classified bottlenecks through reasoning conditions.

To illustrate this approach, we continue our example from the previous section in Figure 6. After classifying four perspective records as bottlenecks, we extract the corresponding workload behaviors from the enriched trace data (labeled as "Rich Trace" in the figure). For instance, the metadata_time

defined in the excessive_metadata_access rule is derived from the original trace data during the enrichment process by using the I/O category field (read, write, or metadata) and the associated time for each category. The identified bottlenecks and enriched trace data are then sent to the rule engine, which applies its rules based on the workload behaviors of the enriched data. For instance, for classified bottleneck dirb, the enriched data showed that the percentage of metadata access on the directory was 73%, out of which the metadata time was dominated by the open operation. This triggered the excessive_metadata_access rule, and that got attached to the bottleneck. Similarly, other rules will be executed with the trace data and potential reasons will be attached to the classified bottlenecks. This example demonstrates the capability of the rule engine to associate reasons as root causes for the classified bottlenecks within WisIO.

4.4 Implementation & APIs

The WisIO tool[1] is implemented in Python for version 3.9 and above. It comprises a core library and two user-facing interfaces. The core library is implemented using the Dask parallel computing library [34] to build a distributed workflow to detect bottlenecks in large-scale performance data. The current implementation can detect I/O bottlenecks from trace data collected using Darshan DXT [47], DFTracer [11], and Recorder [42]. WisIO offers a command-line interface (CLI) and a Python API. The CLI, installable via pip, uses the Hydra configuration framework [35] to configure analysis parameters such as different types of perspectives and the classification threshold. The Python API can be imported into a Jupyter Notebook for interactive workload analysis.

The core library is composed of a task graph which is implemented using Dask DataFrames and Delayed APIs to pipeline and parallelize the tasks across distributed workers. Also, we map partitions of the distributed DataFrame to custom operators that parallelly execute our workflow on the Dask DataFrame. We use memory snapshots of the data structures used within the analysis to improve parallel access across the distributed cluster. This operation applies transformations asynchronously and persists the Dask DataFrame in the distributed memory for future accesses. Finally, we implemented a checkpointing infrastructure to store intermediate stages of our workflow to ensure the recovery and reproducibility of the analysis.

The WisIO tool provides a paginated and human-readable output of all bottlenecks found within the workload. An example of this output is shown in Figure 7. The output has two sections: the workload behavior and the classified bottlenecks with their reasons. The workload behavior provides a

[1]The WisIO tool is open-sourced at https://github.com/grc-iit/wisio. The repository includes usage examples.

```
WisIO
──────────────────── CM1 I/O Characteristics ────────────────────

Runtime          667.81 s
I/O Time         4.12 s - R: 0.05% - W: 14.08% - M: 85.89%
I/O Operations   27.5K ops - R: 4.67% - W: 8.39% - M: 86.95%
I/O Size         21.18 GiB - R: 94.59% - W: 5.41%
Read Requests    4 MiB-16 MiB
Write Requests   4 kiB-16 MiB
Processes/Ranks  1,280 processes
Files            775 files - Shared: 4.90% - FPP: 95.10%
Time Periods     393 time periods
Access Pattern   Sequential: 100.00% - Random: 0.00%

R: Read - W: Write - M: Metadata
──────────────────────────────────────────────────────────────────

──────────────── 75 I/O Bottlenecks with 176 Reasons ──────────────

Time View (23 bottlenecks with 53 reasons)
├── [CR1] 32 processes access 2 files within 1 time period (5)
│   across 32 I/O operations and have an I/O time of 2.19 seconds
│   which is 53.26% of overall I/O time of the workload.
│   └── [Excessive metadata access] Overall 100.00% (2.19 seconds)
│       of I/O time is spent on metadata access, specifically
│       100.00% (2.19 seconds) on the 'open' operation.
└── (22 more)
Process View (24 bottlenecks with 48 reasons)
├── [CR24] 1 process (app1#localhost#600#35184372411024) accesses
│   38 files within 18 time periods across 162 I/O operations and
│   has an I/O time of 3.44 seconds which is 83.66% of overall I/O
│   time of the workload.
│   ├── [Excessive metadata access] Overall 100.00% (3.44 seconds)
│   │   of I/O time is spent on metadata access, specifically
│   │   99.98% (3.44 seconds) on the 'open' operation.
│   └── [Size imbalance] 'read' size is 100.00% (16.00 MiB) of
│       total I/O size.
└── (23 more)
File View (28 bottlenecks with 75 reasons)
├── [CR48] 7 processes access 7 files within 7 time periods across
│   6 I/O operations and have an I/O time of 0.07 seconds which is
│   1.67% of overall I/O time of the workload.
│   ├── [Excessive metadata access] Overall 99.86% (0.07 seconds)
│   │   of I/O time is spent on metadata access, specifically
│   │   99.79% (0.07 seconds) on the 'open' operation.
│   ├── [Operation imbalance] 'write' operations are 16.67% (1
│   │   operations) of total I/O operations.
│   └── [Size imbalance] 'write' size is 100.00% (47.26 kiB) of
│       total I/O size.
└── (27 more)
```

Figure 7: The output produced by WisIO for CM1. I/O characteristics are presented in compact mode. Combinatorial views are excluded for brevity.

quick overview of key characteristics like runtime, I/O time, and operation count, offering insight into the workload's core behavior. The bottleneck classification section lists the top *n* bottlenecks, each with one or more reasons. This example shows the utility and user-driven design of WisIO for application developers and system researchers.

4.5 Enabled Practical Optimizations

WisIO's multi-perspective views detect I/O bottlenecks from different perspectives of the same trace data. This enables holistic optimizations of the workload as it effectively finds critical I/O behavior in different parts of the trace data. For instance, WisIO detects the most problematic files via its per-file perspective. If WisIO reasons that many processes accessing a particular file is a bottleneck, this can be addressed with data layout transformations to make it more amenable to parallel access [2, 9]. If WisIO deems that a small number of files are heavily accessed, it might make sense to cache these files in a burst buffer or node-local

Table 2: Number of I/O operations, files, and processes, and trace size of workloads used in the evaluation.

Workload	Number of			Trace Size
	Ops.	Files	Proc.	
CM1	27.5K	775	1,2K	7.8 MB
Flash	2.5M	1.6K	1K	481 MB
HACC	72.9K	2.5K	1.2K	27.4 MB
Jag (LBANN)	635K	4	5.2K	72.8 GB
Montage (Pegasus)	12.3M	19.6K	11.4K	727 MB
MuMMI (WEMUL)	38.4M	80	74.5K	59.5 GB
CosmoFlow (LBANN)	10.4M	49.5K	652	53.6 GB
1000 Genomes (Pegasus)	646M	21.2M	2.7K	404 GB

storage [39]. For example, the Hermes middleware's "hot-data" policy can be used to place frequently accessed data in faster layers of the hierarchy [10, 24].

Similarly, WisIO can show how different processes contribute to I/O bottlenecks via its per-process perspective. For instance, if WisIO detects that a small number of processes are performing the bulk of the I/O operations, one can implement collective I/O operations to reduce the number of requests sent to the storage system [4, 45]. Similarly, if processes are performing small, non-contiguous I/O operations, one can configure UnifyFS to use different write modes to avoid such overhead [5].

Finally, WisIO can reveal patterns or phases of I/O activity via its timeline perspective. For example, if WisIO detects a phase of intensive I/O, one can apply collective I/O or data sieving, whereas phases write only small amounts of data might require different tuning [9, 10, 24]. Similarly, if WisIO detects metadata operations are causing a bottleneck given a specific time interval, one can explore techniques to distribute metadata operations across multiple servers [16, 25].

5 Evaluation

In this section, we evaluate WisIO's internal components's effectiveness and performance. Then, we compare WisIO against Dristhi. Finally, we employ WisIO to detect bottlenecks for five scientific workloads and present the results.

5.1 Methodology

5.1.1 Testbed. We run the experiments on the Lassen cluster at Lawrence Livermore National Laboratory (LLNL) [18], comprising 795 nodes with two IBM POWER9 CPUs (IBM AC922 servers) and 256GB of system memory.

5.1.2 Software. We chose Recorder [43] to collect I/O traces because it captures metadata operations essential for our bottleneck detection rules.

5.1.3 Scientific Workloads. We evaluate WisIO against real-world HPC workloads with diverse I/O behaviors including deep learning applications and workflows with complex

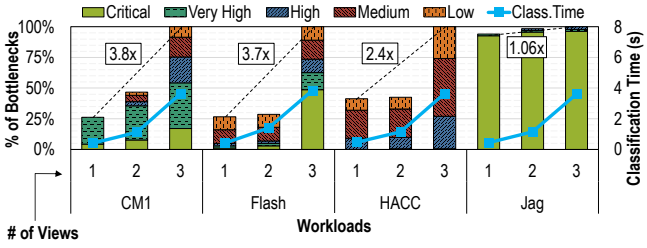


Figure 8: For low-variability workloads, three views yields up to 3.8× more bottlenecks than a single view.

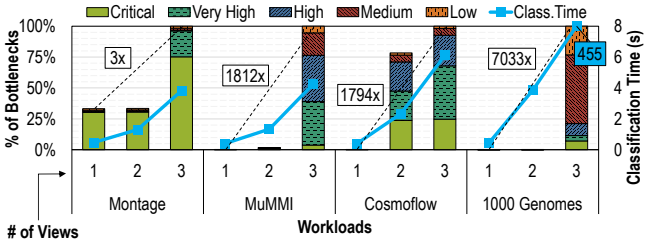


Figure 9: For high-variability workloads, three views yields up to 7033× more bottlenecks than a single view.

data dependencies. These include CM1 [38], Flash [6] (an open radiation MHD simulation code for plasma physics and astrophysics), HACC [17] (a cosmology workload), Jag [37] (a semianalytic AI model of ICF implosions in 3D), Montage [20] (a mosaics-building tool employed in astrophysics), MuMMI [13] (a multi-scale machine-learned modeling infrastructure), CosmoFlow [32] (a deep learning tool for cosmology data analysis), and 1000 Genomes [41]. Table 2 shows the summary of these workloads, including the number of I/O operations, files, and processes found in the I/O traces of the workloads.

5.1.4 Experiment Setup. Each WisIO experiment is conducted across eight distributed nodes using all three views (per-file, per-process, and timeline) with the default threshold value of 45°, unless stated otherwise.

5.2 Internal Evaluation

In this section, we demonstrate the effectiveness and performance of WisIO's internal components. We conducted our evaluations against the workloads listed in Table 2.

5.2.1 Multi-Perspective Views. To understand the impact of multiple perspectives in increasing the bottleneck classification coverage, we run WisIO for IOPS using the 45° threshold with an increasing number of views, from one to three. Specifically, with one view, we focus on the process name perspective. We start with the process name perspective because load imbalance in I/O is a common issue in scientific workflows, and this approach effectively filters out non-problematic data [9]. With two views, we combine

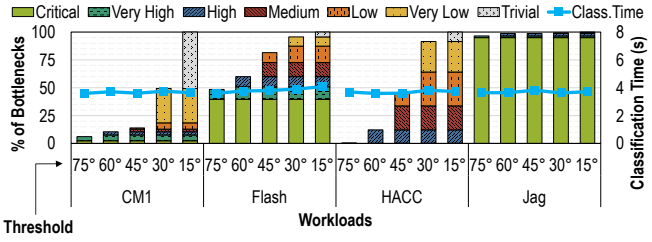


Figure 10: Low-variability workloads exhibit a constant classification time.

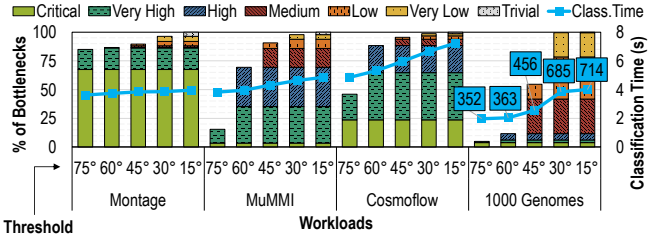


Figure 11: High-variability workloads average a 9% increase in classification time per threshold increase.

the process name and timeline perspectives. Finally, with three views, we incorporate the process name, file name, and timeline perspectives. Throughout our evaluations, we refer to these views as *Process*, *Time*, and *File View*, respectively. We separate the workloads into low- and high-variability workloads due to their unique behavior. Variability refers to the skewness of the number of I/O characteristics per view in the performance data. The results are presented in Figures 8 and 9. In the figures, the x-axis shows the workloads, the primary y-axis shows the percentage of bottlenecks in different classifications, and the secondary y-axis shows the classification time in seconds. The percentage of bottlenecks is based on the maximum number of bottlenecks detected using all three views.

In Figures 8 and 9, we observe that using all three views results in an average of 805× more bottlenecks being classified. Specifically, low-variability workloads show an increase of 1.06–3.8×, while high-variability workloads exhibit a 3–7033× increase compared to using a single view. Adding multiple perspectives for the same trace data improves the bottleneck classification coverage more than the increase in classification time. The trend suggests that as we add more perspective, more behavior will be covered, increasing our bottleneck coverage as well as the cost of analysis.

5.2.2 Impact of Thresholds on Bottleneck Classification. To illustrate the impact of severity parameter thresholds on bottleneck classification, we run WisIO for IOPS with five different threshold values: 15°, 30°, 45°, 60°, and 75°.

The results are presented in Figures 10 and 11. In the figures, the x-axis shows the workloads, the primary y-axis

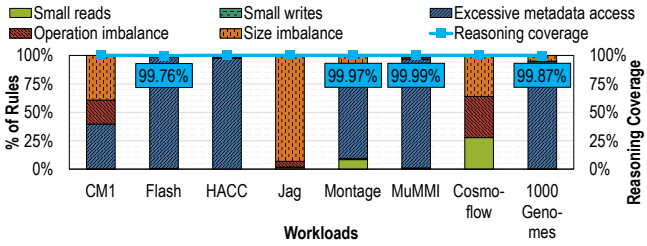


Figure 12: The default rules can reason at least 99.76% of bottlenecks classified.

shows the percentage of bottlenecks in different classifications, and the secondary y-axis shows the classification time in seconds. In Figures 10 and 11, we observe that decreasing the threshold increases the percentage of bottlenecks classified. A threshold of 75° only classifies critical and very high bottlenecks, whereas a threshold of 15° includes even trivial ones. The percentage of bottlenecks is based on the maximum number of bottlenecks detected using the smallest threshold value of 15°. The result shows that lowering the threshold beyond 45° only increases the number of unimportant bottlenecks (low, very low, and trivial). This suggests that although decreasing the threshold detects more bottlenecks, they are generally less critical and lead to longer classification times, indicating a potentially unfavorable trade-off.

5.2.3 Bottleneck Reasoning Coverage. To demonstrate the coverage of our default reasoning rules, we assess the proportion of these rules utilized in identifying the classified bottlenecks, as well as their overall effectiveness in covering all classified bottlenecks. Unlike traditional bottleneck detection tools, our classification algorithm classifies bottlenecks and their severities independently of the reasoning rules (heuristics). Therefore, the reasoning coverage metric specifically measures how well the rules can assign reasons to the already classified bottlenecks, rather than validating root causes. The results are presented in Figure 12. In the figure, the x-axis shows the workloads, the primary y-axis shows the percentage of rules, and the secondary y-axis shows the percentage of reasoning coverage. We observe that WisIO's default rules can reason at least 99.76% of classified bottlenecks. While these results suggest that our five default rules are highly effective in covering most bottlenecks, a small portion remains uncovered. This aligns with our design, as WisIO's metric-driven classification algorithm, being decoupled from heuristics, can classify bottlenecks even when current rules don't provide reasons for edge cases.

5.2.4 Bottleneck Reasoning Performance. To demonstrate the performance of bottleneck reasoning, we evaluate the reasoning throughput against the number of bottlenecks reasoned. The results is presented in Figure 13. In the figure, the x-axis shows the workloads, the primary y-axis shows

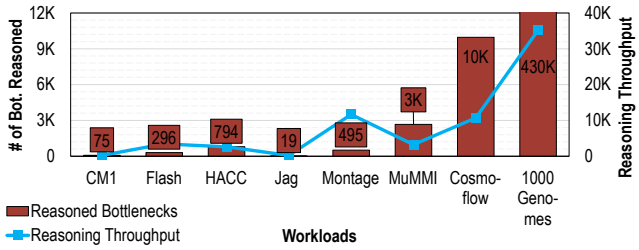


Figure 13: WisIO's bottleneck reasoning can reason 430K bottlenecks with a 35K bottlenecks per second (BPS) throughput, averaging around 8.4K BPS.

the number of bottlenecks reasoned, and the secondary y-axis shows the throughput in bottlenecks per second (BPS). We observe that the reasoning throughput averages around 8.4K BPS. The throughput peaks for 1000 Genomes at 35K BPS. The trend suggests that the reasoning scales well depending on the number of classified bottlenecks.

5.3 Comparison with State-of-the-art Tool

We compare WisIO against Drishti (version 0.6) to study the differences in their performance and coverage. As of writing, Drishti is the only comparable HPC tool due to its similar functionality. We run a microbenchmark that simulates the behavior of an AI application and involves 320 ranks performing 1K read operations (64-256 KB) and one write operation (256 KB to 1 MB) each, resulting in a total data exchange of 39 GB over 7 seconds. Drishti utilizes Darshan's per-file perspective and focuses only on critical bottlenecks. For a fair comparison, we run WisIO with only *File View* enabled and set the threshold to 75° to detect only critical bottlenecks. Both take around 2 seconds to complete the analysis.

The results are presented in Figures 14 and 15. Drishti identifies two critical bottlenecks: a high number of small read requests (P06 in Figure 14) and data transfer imbalance while accessing a shared file (P18 in Figure 14). In contrast, WisIO detects only one bottleneck: the same shared file dominates I/O time, accounting for 82.68% of the overall I/O time (CR1 in Figure 15). WisIO identifies three reasons for this bottleneck. First, there is a high number of read accesses to this file. This is similar to Drishti's P06, but not quite the same, as Drishti examines the entire application's read operations, while WisIO focuses on the performance of this single file. Second, write operations consume a substantial I/O time. Third, write operations are small (under 1 MB) on average. These findings demonstrate that having a high number of read operations does not automatically translate to a bottleneck, underlining the need for decoupled bottleneck detection.

5.4 Use Cases

To assess WisIO's effectiveness, we test it against eight real-world HPC workloads, using a default 45° threshold across

```
▶ [P06] Application issues a high number (320000) of small read
requests (i.e., < 1MB) which represents 100.00% of all read
requests
▶ [P18] Detected data transfer imbalance caused by stragglers
when accessing 1 shared file.
    ↳ Load imbalance of 50.00% detected while accessing
    "file_0-320.bat"
```

Figure 14: Bottlenecks detected by Drishti demonstrate one-to-one mapping between rules and reasons.

```
1 I/O Bottleneck with 3 Reasons

File View (1 bottleneck with 3 reasons)
└── [CR1] 1 file (file_0-320.bat) has an I/O time of 42.70 seconds
    across 1320 I/O operations which is 82.68% of overall I/O time
    of the workload.
    ├── [Operation imbalance] 'read' operations are 75.76% (1,000
    │   operations) of total I/O operations.
    ├── [Small writes] 'write' time is 99.91% (42.67 seconds) of
    │   I/O time.
    └── [Small writes] Average 'write's are 96.97 kiB, which is
        smaller than 1.00 MiB.
```

Figure 15: Bottlenecks detected by WisIO demonstrate decoupled bottleneck detection and reasoning, with each bottleneck having multiple root causes.

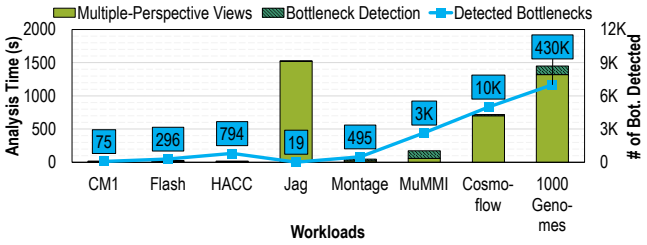


Figure 16: Overall analysis time is dominated by the computation of multi-perspective views, while issue diagnosis takes a fraction of the analysis time.

Process, *File*, and *Time Views*. We record I/O characteristics and detected bottlenecks (Figure 7 shows the output format). For brevity, we gather the important I/O characteristics WisIO reports for all the workloads in Table 3. For the same reason, we only present critical and interesting bottlenecks for five workloads. The analysis time and the number of bottlenecks detected are presented in Figure 16. Notably, the detected bottlenecks represent about 0.2% of the total I/O events across workloads. The distribution of bottleneck reasons and the reasoning coverage are presented in Figure 12.

5.4.1 CM1. simulates atmospheric phenomena like thunderstorms and tornadoes across 193 steps. It uses 16MB configuration files to generate over 750 files, each totaling about 128MB per step. WisIO detects 75 bottlenecks for CM1 and finds 176 reasons for these bottlenecks in 17 seconds with 100% root cause coverage.

We identify four key bottlenecks for CM1. First, 86% of the I/O time is spent on metadata operations. The I/O characteristics panel in Figure 7 shows specifically how much

Table 3: The I/O characteristics WisIO reports for the selected workloads.

Workload	Runtime	I/O Time	I/O Time (%)			I/O Ops.	I/O Ops. (%)			I/O Size	I/O Size (%)	
			R	W	M		R	W	M		R	W
CM1	12 m	4.12 s	0.05	14.08	85.89	27.5K	4.67	8.39	86.95	21.2 GB	94.59	5.41
HACC	32 s	10.37 s	13.48	28.74	57.78	72.9K	17.55	17.55	64.9	1.5 TB	98.04	1.96
Montage (Pegasus)	7 m	3.74 s	83.17	14.56	2.27	12.3M	54.14	44.18	1.68	153.2 GB	74.85	25.15
CosmoFlow (LBANN)	59 m	2.2 m	98.66	0.11	1.36	10.4M	19.87	0.36	79.77	1.4 TB	~100	~0
1000 Genomes (Pegasus)	54 m	22.6 m	1.43	4.53	94.04	646M	81.51	1.66	16.83	71 TB	99.86	0.14

I/O time is spent on metadata operations. Second, every first rank per node (i.e., 32 ranks) reads configuration files during the initialization of the application. Before reading the files, all ranks issue "open" calls concurrently, which causes stagnation on the parallel file system. This bottleneck appears in the *Time View* section of the Bottlenecks panel in Figure 7. Third, 31 ranks except the rank 0 only issue "read" calls. The size imbalance rule accurately detects that these ranks are 100% "read" intensive (see CR24 in Figure 7). Finally, while simulation files are written via small writes (averaging 48 kiB), these small writes do not seem to be the primary issue. Metadata accesses still dominate the total I/O time spent on these files. Specifically, 99.95% of the I/O time is spent on "open" calls (see CR48 in Figure 7).

```
Time View (14 bottlenecks with 36 reasons)
├── [CR1] 164 processes access 164 files within 1 time period (7)
│   across 286 I/O operations and have an I/O time of 4.15 seconds
│   which is 40.04% of overall I/O time of the workload.
│   ├── [Excessive metadata access] Overall 100.00% (4.15 seconds)
│   │   of I/O time is spent on metadata access, specifically
│   │   100.00% (4.15 seconds) on the 'open' operation.
│   ├── [Operation imbalance] 'write' operations are 11.54% (33
│   │   operations) of total I/O operations.
│   └── [Size imbalance] 'write' size is 36.44% (1.01 GiB) of total
│       I/O size.
├── [CR2] 441 processes access 441 files within 1 time period (9)
│   across 677 I/O operations and have an I/O time of 8.44 seconds
│   which is 81.45% of overall I/O time of the workload.
│   ├── [Excessive metadata access] Overall 100.00% (8.44 seconds)
│   │   of I/O time is spent on metadata access, specifically
│   │   100.00% (8.44 seconds) on the 'open' operation.
│   └── [Size imbalance] 'read' size is 100.00% (912.00 MiB) of
│       total I/O size.
└── (12 more)
```

Figure 17: *Time View* reveals bottlenecks within HACC at the 7th and 9th seconds due to checkpoint/restart.

5.4.2 HACC. simulates the universe's evolution using particle-mesh techniques. With 16M input particles, each process writes nine variables, totaling 632 MB per process. The benchmark generates 790GB of data, simulating checkpointing and restart. WisIO detects 794 bottlenecks for HACC and finds 1050 reasons in 17 seconds with 100% root cause coverage.

We identify four key bottlenecks for HACC. First, 58% of the I/O time is spent on metadata operations (see HACC in Table 3). Second, there are 4 times more metadata operations than read and write operations (see HACC in Table 3). Third, 164 processes access 164 files during the 7th second across 286 I/O operations and have an I/O time of 4.15 seconds which is 40% of overall I/O time of the workload. This is the moment when the application checkpoints simulation data for the first time. Although this time period is write-intensive, the I/O time is dominated by open operations (see CR1 in Figure 17). Finally, 441 processes access 441 files within during 9th second across 677 I/O operations and have an I/O time of 8.44 seconds which is 81% of overall I/O time of the workload. This is when the application reads back the checkpointed data. Although this time period is read-intensive, the I/O time is dominated by open operations (see CR2 in Figure 17).

```
Process>App View (7 bottlenecks with 18 reasons)
├── [CR1580] 1 app (mImgtbl) accesses 208 files within 9 time
│   periods across 7,224 I/O operations and has an I/O time of 0.43
│   seconds which is 11.54% of overall I/O time of the workload.
│   ├── [Size imbalance] 'read' size is 99.54% (540.38 kiB) of
│   │   total I/O size.
│   ├── [Small reads] 'read' time is 96.76% (0.42 seconds) of I/O
│   │   time.
│   └── [Small reads] Average 'read's are 76.59800664451828 B,
│       which is smaller than 1.00 MiB.
└── (6 more)
```

Figure 18: *App View* reveals bottlenecks within the Montage workflow, where seven applications spend most of their I/O time on read operations.

5.4.3 Montage (Pegasus). converts sky-survey data from FITS to PNG for survey NGC 3372. It's a six-stage workflow featuring data-parallel mosaic building via 10 cooperating applications. 1024 FITS files are distributed among 32 nodes, each handling 16 files. The top three I/O-intensive applications are *mDiff*, *mBackground*, and *mProject*. WisIO detects 495 bottlenecks for the Montage workflow and finds 1735 reasons in 48 seconds with 99.97% root cause coverage.

We identify two key bottlenecks for Montage. First, the workflow accesses data with a small request size (less than 1 MB). WisIO's I/O characteristics reports that 85.73% read operations are smaller than 4 KB. Second, WisIO's *App View* shows that seven applications within the workflow exhibit this behavior. For instance, CR1580 in Figure 18 shows that the average read operations are 76 B which constitute 96.76% I/O time for the *mImgtbl* application.

```
File>File Pattern View (2 bottlenecks with 2 reasons)
├── [CR38346] 1 process accesses 1 file pattern
│   (sgd.testing.epoch.[0-9].step.[0-9]_layer[0-9]_output[0-9].csv)
│   within 141 time periods across 420 I/O operations and has an
│   I/O time of 0.63 seconds which is 0.47% of overall I/O time of
│   the workload.
│   └── [Excessive metadata access] Overall 100.00% (0.63 seconds)
│       of I/O time is spent on metadata access, specifically
│       86.43% (0.54 seconds) on the 'open' operation.
└── [CR38347] 1 process accesses 1 file pattern
    (sgd.testing.epoch.[0-9].step.[0-9]_cosmoflow_module[0-9]_fc[0…
    within 140 time periods across 420 I/O operations and has an
    I/O time of 0.37 seconds which is 0.28% of overall I/O time of
    the workload.
    └── [Excessive metadata access] Overall 100.00% (0.37 seconds)
        of I/O time is spent on metadata access, specifically
        75.36% (0.28 seconds) on the 'open' operation.
```

Figure 19: *File Pattern View* shows two file patterns are dominated by metadata operations within CosmoFlow.

```
Process>App View (4 bottlenecks with 9 reasons)
├── [CR858] 1 app (pegasus-kickstart) has an I/O time of 8.34
│   seconds across 46 I/O operations which is 0.61% of overall I/O
│   time of the workload.
│   └── [Excessive metadata access] No reason found, investigation
│       needed!
├── [CR859] 1 app (mutation_overlap) has an I/O time of 69.29
│   seconds across 1,255,334 I/O operations which is 5.10% of
│   overall I/O time of the workload.
│   ├── [Excessive metadata access] Overall 63.95% (44.31 seconds)
│   │   of I/O time is spent on metadata access, specifically
│   │   62.71% (43.45 seconds) on the 'open' operation.
│   ├── [Small writes] 'write' time is 65.72% (45.53 seconds) of
│   │   I/O time.
│   └── [Small writes] Average 'write's are 32.70 kiB, which is
│       smaller than 1.00 MiB.
└── (2 more)
```

Figure 20: *App View* reveals unseen behavior within the 1000 Genomes workflow, showcasing WisIO's decoupled classification ability.

5.4.4 CosmoFlow (LBANN). utilizes deep learning to estimate critical cosmological parameters from 3D simulations. It deals with a dataset consisting of 10K simulated universes with four redshifts and 5123 voxels, stored as 16-bit integers. The dataset, totaling 1.5TB, comprises 50K samples of 32MB each in HDF5 format. WisIO detects 10K bottlenecks for CosmoFlow and finds 39K reasons for these bottlenecks in 12 minutes with 100% root cause coverage.

We identify two key bottlenecks for CosmoFlow. First, the application's I/O time is dominated by read operations (see CosmoFlow in Table 3). Second, WisIO's *File Pattern View* shows that the I/O time for two file patterns across 420 I/O operations are dominated by metadata operations (CR38346 and CR38347 in Figure 19).

5.4.5 1000 Genomes (Pegasus). is a data-intensive bioinformatics workflow that computes human genome mutation overlaps and processes a specified number of chromosomes in parallel [14]. WisIO reports that the workflow has 7 cooperation applications. The top three I/O-intensive applications are *individuals*, *frequency*, and *individuals_merge*. WisIO detects 430K bottlenecks for the 1000 Genomes workflow and finds 560K reasons for these bottlenecks in 24 minutes with 99.87% root cause coverage.

We identify three key bottlenecks for 1000 Genomes. First, the *individuals* application with the workflow spends significant time performing I/O operations [14]. WisIO's I/O characteristics reports that this application reads 65 TB data and writes 5 GB data via 556M I/O operations (86% of total I/O operations). Second, the *pegasus-kickstart* spends significant time performing metadata operations (CR858 in Figure 20). However, initially it can't find a specific reason for this particular bottleneck. Further investigation via the bottleneck inspect command shows that the application performs *access* operations during the initialization of the workflow, causing the issue. By default, the *access* operations are not covered by WisIO's default rules. We resolve this by including a rule that considers access_time via WisIO's rule_definitions configuration. Finally, the *mutation_overlap* application performs 1.2M I/O operations which constitute 5.1% of overall I/O time. These operations take 70 seconds in total and 63% of this time is spent on *open* operations. Additionally, the application mostly performs small writes, averaging around 32 KB (CR859 in Figure 20).

6 Conclusion

WisIO offers an automated bottleneck detection tool for large-scale HPC workflows. WisIO enables parallel and distributed analysis of multi-terabyte-scale performance data, examines it from multiple perspectives, uses metric-driven bottleneck classification, and allows extensible mapping of bottlenecks to root causes. WisIO's multi-perspective approach detects up to 805× more bottlenecks compared to analyzing performance data from only one perspective. WisIO's decoupled and metric-driven classification algorithm can identify unseen bottlenecks in HPC workflows with the throughput of 340K bottlenecks per second. WisIO's reasoning engine utilizes heuristics to map workflow behaviors to classified bottlenecks with the throughput of 35K bottlenecks per second. Finally, we validate WisIO's effectiveness with large-scale performance data from real-world HPC workflows against state-of-the-art solutions and show its 11× faster performance and ability to detect up to 144× more bottlenecks.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the DOE Early Career Research Program (LLNL-CONF-862440). Also, this research is supported in part by the National Science Foundation (NSF) under Grants OAC-2104013, OAC-2313154, and OAC-2411318.

References

[1] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. 2018. Optimizing I/O Performance of HPC Applications with Autotuning. *ACM Transactions on Parallel Computing* 5, 4 (Dec. 2018), 1–27. https://doi.org/10.1145/3309205

[2] Jean Luca Bez, Suren Byna, and Shadi Ibrahim. 2024. I/O Access Patterns in HPC Applications: A 360-Degree Survey. *Comput. Surveys* 56, 2 (Feb. 2024), 1–41. https://doi.org/10.1145/3611007

[3] Jean Luca Bez, Lawrence Berkeley National Laboratory, Hammad Ather, Lawrence Berkeley National Laboratory, Suren Byna, and Lawrence Berkeley National Laboratory. 2022. Drishti: Guiding End-Users in the I/O Optimization Journey. (2022).

[4] Jean Luca Bez, Houjun Tang, Bing Xie, David Williams-Young, Rob Latham, Rob Ross, Sarp Oral, and Suren Byna. 2021. I/O Bottleneck Detection and Tuning: Connecting the Dots using Interactive Log Analysis. In *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*. IEEE, St. Louis, MO, USA, 15–22. https://doi.org/10.1109/PDSW54622.2021.00008

[5] Michael J. Brim, Adam T. Moody, Seung-Hwan Lim, Ross Miller, Swen Boehm, Cameron Stanavige, Kathryn M. Mohror, and Sarp Oral. 2023. UnifyFS: A User-level Shared File System for Unified Access to Distributed Local Storage. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, St. Petersburg, FL, USA, 290–300. https://doi.org/10.1109/IPDPS54959.2023.00037

[6] A.C. Calder, B.C. Curts, L.J. Dursi, B. Fryxell, G. Henry, P. MacNece, K. Olson, P. Ricker, R. Rosner, F.X. Timmes, H.M. Tufo, J.W. Truran, and M. Zingale. 2000. High-Performance Reactive Fluid Flow Simulations Using Adaptive Mesh Refinement on Thousands of Processors. In *ACM/IEEE SC 2000 Conference (SC'00)*. IEEE, Dallas, TX, USA, 56–56. https://doi.org/10.1109/SC.2000.10010

[7] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 Characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, New Orleans, LA, USA, 1–10. https://doi.org/10.1109/CLUSTR.2009.5289150

[8] Fahim Chowdhury, Yue Zhu, Francesco Di Natale, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Weikuan Yu. 2020. Emulating I/O Behavior in Scientific Workflows on High Performance Computing Systems. In *2020 IEEE/ACM Fifth International Parallel Data Systems Workshop (PDSW)*. IEEE, GA, USA, 34–39. https://doi.org/10.1109/PDSW51947.2020.00011

[9] Hariharan Devarajan and Kathryn Mohror. 2022. Extracting and characterizing I/O behavior of HPC workloads. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Heidelberg, Germany, 243–255. https://doi.org/10.1109/CLUSTER51413.2022.00037

[10] Hariharan Devarajan and Kathryn Mohror. 2023. Mimir: Extending I/O Interfaces to Express User Intent for Complex Workloads in HPC. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, St. Petersburg, FL, USA, 178–188. https://doi.org/10.1109/IPDPS54959.2023.00027

[11] Hariharan Devarajan, Loïc Pottier, Kaushik Velusamy, Huihuo Zheng, Izzet Yildirim, Olga Kogiou, Weikuan Yu, Anthony Kougkas, Xian-He Sun, Jae Seung Yeom, and Kathryn Mohror. 2024. DFTracer: An Analysis-Friendly Data Flow Tracer for AI-Driven Workflows. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Atlanta, GA, USA, 1–24. https://doi.org/10.1109/SC41406.2024.00023

[12] Hariharan Devarajan, Huihuo Zheng, Anthony Kougkas, Xian-He Sun, and Venkatram Vishwanath. 2021. DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, Melbourne, Australia, 81–91. https://doi.org/10.1109/CCGrid51090.2021.00018

[13] Francesco Di Natale, Harsh Bhatia, Timothy S. Carpenter, Chris Neale, Sara Kokkila-Schumacher, Tomas Oppelstrup, Liam Stanton, Xiaohua Zhang, Shiv Sundram, Thomas R. W. Scogland, Gautham Dharuman, Michael P. Surh, Yue Yang, Claudia Misale, Lars Schneidenbach, Carlos Costa, Changhoan Kim, Bruce D'Amora, Sandrasegaram Gnanakaran, Dwight V. Nissley, Fred Streitz, Felice C. Lightstone, Peer-Timo Bremer, James N. Glosli, and Helgi I. Ingólfsson. 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*. ACM, Denver Colorado, 1–16. https://doi.org/10.1145/3295500.3356197

[14] Tu Mai Anh Do, Loic Pottier, Orcun Yildiz, Karan Vahi, Patrycja Krawczuk, Tom Peterka, and Ewa Deelman. 2022. Accelerating Scientific Workflows on HPC Platforms with In Situ Processing. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, Taormina, Italy, 1–10. https://doi.org/10.1109/CCGrid54584.2022.00009

[15] Bin Dong, Jean Luca Bez, and Suren Byna. 2023. AIIO: Using Artificial Intelligence for Job-Level and Automatic I/O Performance Bottleneck Diagnosis. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*. ACM, Orlando FL USA, 155–167. https://doi.org/10.1145/3588195.3592986

[16] Peter Harrington. 2018. Diagnosing Parallel I/O Bottlenecks in HPC Applications. (2018). https://api.semanticscholar.org/CorpusID:52256719

[17] Katrin Heitmann, Thomas D. Uram, Hal Finkel, Nicholas Frontiere, Salman Habib, Adrian Pope, Esteban Rangel, Joseph Hollowed, Danila Korytov, Patricia Larsen, Benjamin S. Allen, Kyle Chard, and Ian Foster. 2019. HACC Cosmological Simulations: First Data Release. *The Astrophysical Journal Supplement Series* 244, 1 (Sept. 2019), 17. https://doi.org/10.3847/1538-4365/ab3724

[18] HPC @ LLNL. [n. d.]. Lassen. https://hpc.llnl.gov/hardware/compute-platforms/lassen

[19] Mihailo Isakov, Eliakin Del Rosario, Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert B. Ross, and Michel A. Kinsy. 2020. HPC I/O Throughput Bottleneck Analysis with Explainable Local Models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Atlanta, GA, USA, 1–13. https://doi.org/10.1109/SC41405.2020.00037

[20] Joseph C. Jacob, Daniel S. Katz, G. Bruce Berriman, John C. Good, Anastasia C. Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei Hui Su, Thomas A. Prince, and Roy Williams. 2009. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering* 4, 2 (2009), 73. https://doi.org/10.1504/IJCSE.2009.026999

[21] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, and Yongseok Son. 2023. Design and implementation of I/O performance prediction scheme on HPC systems through large-scale log analysis. *Journal of Big Data* 10, 1 (May 2023), 65. https://doi.org/10.1186/s40537-023-00741-4

[22] Andreas Knüpfer, Christian Rössel, Dieter An Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011*, Holger Brunst, Matthias S. Müller, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–91. https://doi.org/10.1007/978-3-642-31476-6_7

[23] Olga Kogiou, Hariharan Devarajan, Chen Wang, Weikuan Yu, and Kathryn Mohror. 2023. I/O characterization and performance evaluation of large-scale storage architectures for heterogeneous workloads. In *2023 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*. IEEE, Santa Fe, NM, USA, 44–45. https://doi.org/10.1109/CLUSTERWorkshops61457.2023.00017

[24] 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*. ACM, Tempe Arizona, 219–230. https://doi.org/10.1145/3208040.3208059

[25] Samuel Lang, Philip Carns, Robert Latham, Robert Ross, Kevin Harms, and William Allcock. 2009. I/O performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, Portland Oregon, 1–12. https://doi.org/10.1145/1654059.1654100

[26] Glenn K. Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J. Wright. 2018. A Year in the Life of a Parallel File System. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Dallas, TX, USA, 931–943. https://doi.org/10.1109/SC.2018.00077

[27] Glenn K Lockwood, Nicholas J Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. 2018. TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis. *Proceedings of the 2018 Cray User Group* (2018).

[28] Glenn K. Lockwood, Wucherl Yoo, Suren Byna, Nicholas J. Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. 2017. UMAMI: a recipe for generating meaningful metrics through holistic I/O performance analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems - PDSW-DISCS '17*. ACM Press, Denver, Colorado, 55–60. https://doi.org/10.1145/3149393.3149395

[29] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. 2010. Managing Variability in the IO Performance of Petascale Storage Systems. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, New Orleans, LA, USA, 1–12. https://doi.org/10.1109/SC.2010.32

[30] Huong Luu, Babak Behzad, Ruth Aydt, and Marianne Winslett. 2013. A multi-level approach for understanding I/O activity in HPC applications. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Indianapolis, IN, USA, 1–5. https://doi.org/10.1109/CLUSTER.2013.6702690

[31] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. 2015. A Multiplatform Study of I/O Behavior on Petascale Supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, Portland Oregon USA, 33–44. https://doi.org/10.1145/2749246.2749269

[32] Amrita Mathuriya, Deborah Bard, Peter Mendygral, Lawrence Meadows, James Arnemann, Lei Shao, Siyu He, Tuomas Karna, Diana Moise, Simon J. Pennycook, Kristyn Maschhoff, Jason Sewall, Nalini Kumar, Shirley Ho, Michael F. Ringenburg, Prabhat Prabhat, and Victor Lee. 2018. CosmoFlow: Using Deep Learning to Learn the Universe at Scale. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Dallas, TX, USA, 819–829. https://doi.org/10.1109/SC.2018.00068

[33] Open-source. 2015. Darshan-util. https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html

[34] Open-source. 2015. Dask. https://www.dask.org/

[35] Open-source. 2019. Hydra. https://hydra.cc/

[36] Arnab K. Paul, Olaf Faaland, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Ali R. Butt. 2020. Understanding HPC Application I/O Behavior Using System Level Statistics. In *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, Pune, India, 202–211. https://doi.org/10.1109/HiPC50609.2020.00034

[37] J. Luc Peterson, Ben Bay, Joe Koning, Peter Robinson, Jessica Semler, Jeremy White, Rushil Anirudh, Kevin Athey, Peer-Timo Bremer, Francesco Di Natale, David Fox, Jim A. Gaffney, Sam A. Jacobs, Bhavya Kailkhura, Bogdan Kustowski, Steven Langer, Brian Spears, Jayaraman Thiagarajan, Brian Van Essen, and Jae-Seung Yeom. 2022. Enabling machine learning-ready HPC ensembles with Merlin. *Future Generation Computer Systems* 131 (June 2022), 255–268. https://doi.org/10.1016/j.future.2022.01.024

[38] Hafizur Rahman, Michel M. Verstraete, and Bernard Pinty. 1993. Coupled surface-atmosphere reflectance (CSAR) model: 1. Model description and inversion on synthetic data. *Journal of Geophysical Research* 98, D11 (1993), 20779. https://doi.org/10.1029/93JD02071

[39] Robert Ross, Lee Ward, Philip Carns, Gary Grider, Scott Klasky, Quincey Koziol, Glenn K. Lockwood, Kathryn Mohror, Bradley Settlemyer, and Matthew Wolf. 2018. *Storage Systems and Input/Output: Organizing, Storing, and Accessing Data for Scientific Discovery. Report for the DOE ASCR Workshop on Storage Systems and I/O. [Full Workshop Report]*. Technical Report 1491994. 1491994 pages. https://doi.org/10.2172/1491994

[40] Frank Schmuck and Roger Haskin. 2002. GPFS: A Shared-Disk file system for large computing clusters. In *Conference on file and storage technologies (FAST 02)*.

[41] The 1000 Genomes Project Consortium. 2010. A map of human genome variation from population-scale sequencing. *Nature* 467, 7319 (Oct. 2010), 1061–1073. https://doi.org/10.1038/nature09534

[42] 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, New Orleans, LA, USA, 1–8. https://doi.org/10.1109/IPDPSW50202.2020.00176

[43] Chen Wang, Izzet Yildirim, Hariharan Devarajan, Kathryn Mohror, and Marc Snir. 2025. Recorder: Comprehensive Parallel I/O Tracing and Analysis. *arXiv preprint* arXiv:2501.04654 (2025). https://doi.org/10.48550/ARXIV.2501.04654

[44] Teng Wang, Suren Byna, Glenn K. Lockwood, Shane Snyder, Philip Carns, Sunggon Kim, and Nicholas J. Wright. 2019. A Zoom-in Analysis of I/O Logs to Detect Root Causes of I/O Performance Bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, Larnaca, Cyprus, 102–111. https://doi.org/10.1109/CCGRID.2019.00021

[45] Teng Wang, Shane Snyder, Glenn Lockwood, Philip Carns, Nicholas Wright, and Suren Byna. 2018. IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Belfast, 466–476. https://doi.org/10.1109/CLUSTER.2018.00062

[46] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. 2012. Characterizing output bottlenecks in a supercomputer. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Salt Lake City, UT, 1–11. https://doi.org/10.1109/SC.2012.28

[47] Cong Xu, Shane Snyder, Omkar Kulkarni, Vishwanath Venkatesan, Philip Carns, Suren Byna, Robert Sisneros, and Kalyana Chadalavada. 2017. DXT: Darshan eXtended Tracing. (2017).

[48] Izzet Yildirim, Hariharan Devarajan, Anthony Kougkas, Xian-He Sun, and Kathryn Mohror. 2023. IOMax: Maximizing Out-of-Core I/O Analysis Performance on HPC Systems. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. ACM, Denver CO USA, 1209–1215. https://doi.org/10.1145/3624062.3624191

[49] Zhaobin Zhu, Niklas Bartelheimer, and Sarah Neuwirth. 2023. An Empirical Roofline Model for Extreme-Scale I/O Workload Analysis. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, St. Petersburg, FL, USA, 622–627. https://doi.org/10.1109/IPDPSW59300.2023.00106

[50] Zhaobin Zhu and Sarah Neuwirth. 2023. Characterization of Large-scale HPC Workloads with non-naïve I/O Roofline Modeling and Scoring. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, Ocean Flower Island, China, 737–744. https://doi.org/10.1109/ICPADS60453.2023.00112