Performance Analysis for Complex Scientific Workflows

Aliza Lisan

Department of Computer Science
University of Oregon
Eugene OR, USA
alisan@uoregon.edu

***Abstract*—Although bulk-synchronous high-performance computing (HPC) applications have long been the dominant approach within scientific computing on supercomputers, complex scientific workflows are increasingly popular. Complex scientific workflows are workflows that span multiple tasks, and often include heterogeneous architectures and/or the integration of AI and machine learning. The increasing prevalence of complex scientific workflows requires fresh investigation of performance analysis and visualization approaches that were originally designed for single, tightly coupled applications. This survey examines the current ecosystem of HPC performance and visualization tools, highlighting design trade-offs between accuracy, overhead, scalability, and usability. It categorizes tools by methodological choices such as instrumentation versus sampling and profile versus trace analysis, while assessing their capability for capturing both inter- and intra-application performance behavior. The survey also discusses Workflow Management Systems (WMS) in terms of workflow orchestration, performance measurement, and visualization capabilities. Finally, it also reviews visualization frameworks ranging from low-level profilers to holistic workflow visualizers, highlighting gaps in the integration of data from disparate sources. Collectively, these insights highlight the need for workflow-aware visualization and analysis tools that enable coherent, fine-, and coarse-grained understanding of performance in emerging complex scientific computing workflows.**

***Index Terms*—Scientific workflows, AI/ML-enabled workflows, performance analysis, visualization**

I. Introduction

HPC has been one of the most active research areas over the last 40 years [1]. High-performance computing (HPC) bulk-parallel applications execute a single program simultaneously on many processors. These applications, often built using programming models such as MPI, have underpinned scientific computing for decades. These bulk-parallel applications dominated large-scale HPC facilities, with a focus on scalability of a single binary over many cores. However, with the growing use of AI/ML in software and heterogeneity in hardware, these monolithic applications are increasingly complemented by scientific workflows.

Scientific workflows let scientists model data processing tasks and their dependencies as directed acyclic graphs (DAGs), where nodes represent tasks and edges represent data flows [2]. HPC users can embed ML training, inference, and analytics within scientific workflows, often running on heterogeneous infrastructure or in cloud/HPC hybrid setups, hence increasing their complexity. These complex scientific workflows explore a wide range of parameters, utilize distributed data setups, and consist of multiple steps of interoperating tasks. As this complexity increases, understanding and improving how efficiently such workflows use underlying architectural resources becomes crucial. Consequently, the scope of performance analysis, the study of how well applications utilize computational resources, should expand from focusing primarily on bulk-parallel applications to address the diverse behaviors and multistage structure of scientific workflows [3].

Modern supercomputers are extremely costly to operate, and their performance can degrade significantly without careful tuning, making inefficient execution both wasteful and scientifically limiting. Because of this, performance analysis tools are critical in HPC since they enable developers and scientists to achieve high utilization and ensure that scientific codes run efficiently on large-scale HPC and cloud platforms. They enable users and developers to understand how HPC applications interact with system hardware, reveal hidden inefficiencies, and guide optimization decisions across compute, memory, communication, and I/O layers. As HPC systems become more complex, with heterogeneous processors, deep memory hierarchies, and massive parallelism, these tools become indispensable in capturing scalable and detailed performance data.

There exists a wide variety of performance analysis tools tailored to different needs; some specialize in I/O behavior, others focus on CPU/memory profiling, and many provide multiple types of performance data. These tools differ in capability (e.g., sampling vs. full instrumentation, profiling vs. tracing, hardware counter monitoring vs. software event logging) and target different layers of the system. Some of the tools have their own analysis and visualization features, while others generate data in formats which can be ingested by existing visualization tools. However, most of the performance tools were originally developed with the performance analysis of a single bulk-parallel application or simulation in mind.

As scientific workflows grow in complexity, featuring heterogeneous tasks, dynamic data flows, and distributed execution across HPC and cloud platforms, the requirements placed on performance analysis tools are evolving. Many existing tools were designed for tightly coupled bulk-parallel applica-

tions and may struggle to accommodate the data movement, task orchestration, and multi-component dependencies found in modern complex workflows. Consequently, performance tools must now support richer instrumentation, cross-task correlation, heterogeneous execution models, and scalable end-to-end analysis and visualization beyond single-application runs.

This paper presents a survey of the state-of-the-art in performance analysis, evaluation, and visualization tools for large-scale parallel applications and scientific workflows running on HPC systems. By combining insights from these areas, the paper proposes a taxonomy of performance analysis and visualization tools, which constitutes its core contribution. In a nutshell, this paper contributes the following:

- Description of bulk-synchronous HPC applications and their characteristic (Section II).
- Review of the nature of complex scientific workflows, their performance analysis requirements, and Workflow Management Systems (section III).
- Survey of state-of-the-art performance analysis tools for bulk-parallel HPC applications and examination of whether they are "workflow aware" (Section IV).
- Survey of existing performance visualization tools and discussion of their ability to handle workflow-level performance data from disparate sources (Section V).

Lastly, section VI concludes this survey. It also discusses emerging research directions in which Large Language Models (LLMs) are applied for performance prediction in order to guide users towards optimal workflow configurations.

II. Bulk-synchronous HPC Applications

Bulk-synchronous HPC applications adhere to the Bulk Synchronous Parallel (BSP) model, a structured approach to parallel computing, published in 1990 by Leslie Valiant. BSP is a parallel computing model used for the design and implementation of parallel algorithms and applications [4]. In the BSP model, the computation is organized into a sequence of *supersteps*. The processors proceed through each superstep asynchronously, while synchronization is performed by the barrier at the end of each superstep–hence the name of "bulk-synchronous" parallelism [5], [6]. Each superstep comprises three distinct phases. 1) **Local computations**: Each processor performs computations using its local data. 2) **Communication**: Processors exchange data with others as needed. 3) **Synchronization**: A barrier synchronization ensures that all processors have completed their computations and communications before proceeding to the next superstep. This model simplifies the design and analysis of parallel algorithms by providing clear synchronization points, which help manage data dependencies and communication overhead. Figure 1 shows a similar representation of the BSP model.

Bulk-parallel HPC applications are particularly prevalent in fields requiring large-scale data processing and complex computations. Common examples include:

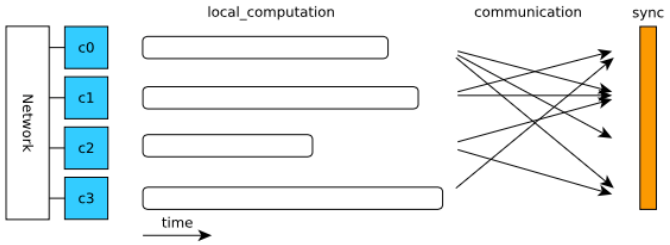


Fig. 1. Illustration of a superstep for the bulk-synchronous parallel model: processing, communication, and global synchronization [7].

- **Scientific Simulations**: Modeling physical phenomena such as climate patterns, astrophysical events, or molecular interactions.
- **Graph Processing**: Analyzing large networks in social sciences, biology, or computer science, where operations such as traversal and shortest path calculations are common.
- **Data Analytics**: Processing vast datasets to extract meaningful insights, often involving operations like sorting, searching, and statistical computations.

The BSP model's structured approach allows these applications to scale efficiently across numerous processors, maintaining performance predictability, and simplifying the management of inter-processor communications.

With the evolution of computer architectures (*multi-core* and *many-core*), parallel computing has also evolved from the theoretical models such as BSP and Parallel Random Access Machine (PRAM) to today's advanced multi-core and heterogeneous architectures. Although BSP provided a foundational structure for parallel computing, contemporary applications have diversified in complexity, architecture, and execution models. As computational problems grew in complexity and hardware architectures became more heterogeneous, BSP's limitations, including its rigid synchronization barriers, became apparent.

Traditionally, most scientific applications are implemented using the Message-Passing Interface (MPI), often coupled with another model for threaded on-node execution. The other models MPI is coupled with include OpenMP for multi-core architectures and CUDA and OpenACC for GPU architectures. However, in order to achieve improved performance, productivity, or both on exascale systems, new and emerging programming models have been introduced, including Chapel, Charm++, Liszt, and Loci. Karlin *et al.* [8] compare the newer programming approaches with the traditional MPI implementation and found that Loci and Chapel result in programs that are up to 80% smaller based on source lines of code (SLOC).

The following list points out some of the characteristics implemented to augment bulk-synchronous parallel applications [9].

- **Heterogeneous Computing**: Modern applications often leverage a mix of CPUs, GPUs, and specialized accelerators. This heterogeneity allows for optimized perfor-

mance across diverse tasks, but introduces complexity in workload distribution and memory management.
- **Asynchronous Execution**: To mitigate the bottlenecks of global synchronization, many applications adopt asynchronous models. This approach enhances scalability and performance, especially in environments with variable processing times.
- **Dynamic Load Balancing**: Given the irregular nature of many modern computational problems, dynamic load balancing ensures efficient resource utilization by redistributing workloads in real-time based on current system states.
- **Task-Based Parallelism**: Instead of dividing data among processors, task-based models focus on decomposing the problem into discrete tasks.
- **Integration with Machine Learning**: The rise of AI and machine learning has influenced parallel application design. For example, the Elastic BSP [10] model adapts the traditional BSP framework to better suit the iterative and data-intensive nature of distributed deep learning.

Key Takeaways

The Bulk Synchronous Parallel (BSP) model provides a structured foundation for scalable parallel computing through well-defined phases of computation, communication, and synchronization. It remains central to many scientific applications, particularly in simulation and data analytics. As architectures and workloads have evolved, applications have incorporated heterogeneous computing, asynchronous execution, and task-based parallelism for better scalability and flexibility, driving the need for advanced performance analysis methodologies.

III. Scientific Workflows

A scientific workflow can be described as a process designed to accomplish a scientific objective by defining a set of tasks and their dependencies [11]–[15]. Al-Saadi *et al.* describe workflows by grouping them into four motifs: single simulations, ensembles, analysis (experiment driven), and dynamic workflows (execution characteristics are not established a priori) [16]. Workflows are commonly represented in the form of Directed Acyclic Graphs (DAGs), where vertices represent computational tasks, while edges represent data dependencies and flows between the tasks. Scientific workflows aim to accelerate scientific discovery in many ways, including by workflow *Automation*, *Scaling*, *Adoption*, and *Provenance* support, or ASAP for short [17]. Compute-intensive workflows are often based on computational science simulations including, but not limited to, running climate and ocean models, simulations based on physics, chemistry, astronomy, biology, etc. Unlike traditional HPC applications, Workflow Management Systems (WMS) are used to run the disparate components of workflows seamlessly on distributed resources and provide a communication channel between these components. WMSs provide scientists with all the necessary tools required to define, execute, publish, and document their workflows. Some examples of complex compute-intensive workflows include Multiscale Machine-Learned Modeling Infrastructure (MuMMI) [18], American Heart Association Molecule Screening (AHA MoleS) [19], Autonomous MultiScale Library (AMS) [20], ICECap [21], CyberShake [22], Galactic [23], and GUIDE [24].

Data-intensive workflows are commonly used in many data-driven disciplines today, using diverse data sources and distributed computing platforms [14], [25], [26]. Workflows are a systemic way of defining the needed methods and provide an interface between domain scientists and computing infrastructures. With diversity in all disciplines and the large increase in data sources, workflows play an important role in extracting meaningful information from multiple data sources and exploiting a wide range of computational resources [27].

Traditionally, simulation workflows consist of three basic steps. First, the instructions are read from the disk or created from the beginning. Second, the simulation, which is usually some type of stepwise physics code, runs. Third, the results are written back to the disk. In some cases, check pointing or profiling maybe happening and in that case simulations spend a considerable time writing large performance data to disk [3]. Generally, the model remains the same, where a simulation reads, writes one or more times, and terminates.

Figure 2 shows some basic components of a workflow based on data partitioning and aggregation. Usually, several of these structures or components are composed into complex workflows by scientific communities. The Workflow Patterns Initiative [28] identifies different workflow patterns based on control-flow, data, resource, and exception handling perspectives. The basic structure is the **process** structure which takes some data as input, works on it, and produces an output. A sequential combination of several such *processes* forms a **pipeline**, where the output of a process is taken as input by the other process. **Data distribution** or data partitioning jobs either produce data as output or divide larger data into subsets that multiple other jobs process. **Data aggregation** jobs process the outputs of several other jobs and produce the resulting data. The resulting data is often redistributed by **data redistribution** jobs, which act as a synchronization point. *Data distribution* jobs increase **parallelism** in a workflow, while **data aggregation** may represent a reduction in parallelism [29].

A. AI/ML-enabled HPC Workflows

Recently, scientists have started leveraging machine learning capabilities in their workflows, which has lead to a new category of workflows, called as scientific ML (SciML) workflows or AI/ML-enabled workflows [30]. Introduction of Artificial Intelligence (AI) and Machine Learning (ML) in HPC has enabled highly accurate modeling with reduced computational needs. Many workflows now integrate ML models to guide analysis, use specialized computing hardware, and directly integrate simulation, data collection, and training. [31]–[33]. Figure 3 shows the three different modes in which ML can

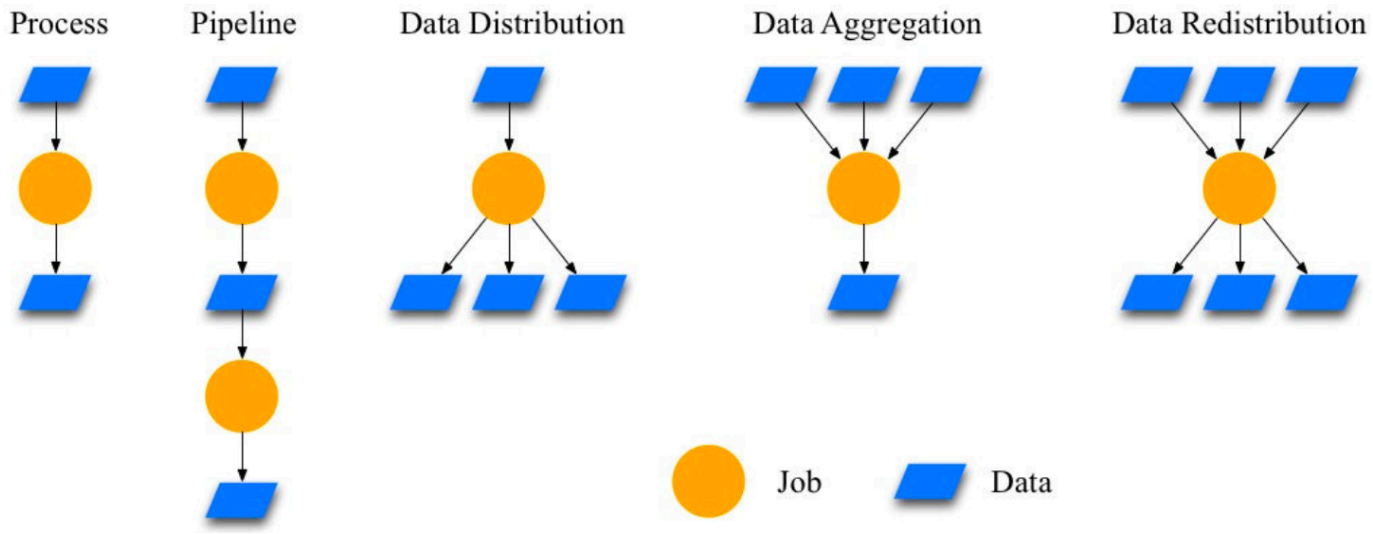


Fig. 2. The five basic structures in scientific workflows based on data partitioning and aggregation [29].

be integrated in HPC codes: **ML-in-HPC**, **ML-out-HPC**, and **ML-about-HPC**.

ML-in-HPC is the scenario where the ML model replaces a computational component in an HPC simulation or possibly the complete computation itself, i.e., ML model serves as a "surrogate." ML-out-HPC represents the situation where the ML model resides "outside" of the traditional HPC simulation loop, but it controls the progression of the HPC workflow dynamically. Lastly, ML-about-HPC denotes a situation where ML models are coupled to main HPC tasks and operate concurrently. The combined usage of all three modes, called the "learning everywhere" paradigm, can lead to **effective performance** for the workflow. Effective performance can be measured in terms of reduced computational cost, time-to-solution, or the achievement of the scientific objective of the workflow [34]. However, AI/ML-enabled workflows often require large quantities of training data, multi-node tasks, and heterogeneous hardware such as CPUs, GPUs, and TPUs.

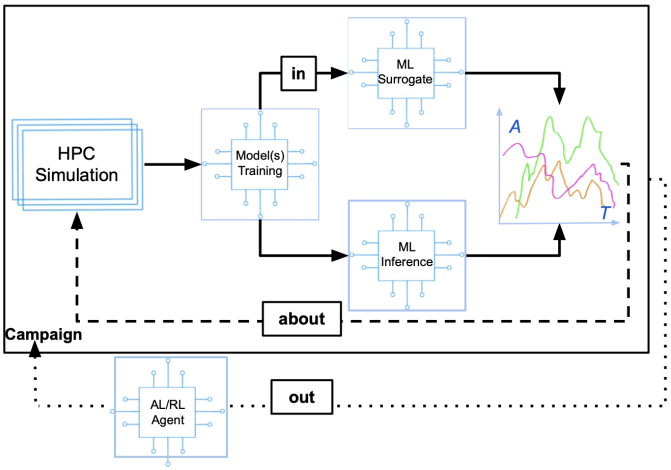


Fig. 3. Three modes of integrating ML in HPC: (1) ML-in-HPC: surrogate models are used to replace part(s) of or the complete simulation. (2) ML-out-HPC: An external AI/ML algorithm (active or reinforcement learning) is used to dynamically control the campaign, or steer the complete workflow. (3) ML-about-HPC: AI/ML complements traditional computational tasks, improving their results or efficiency [34].

B. Difference between Traditional and AI/ML Workflows

Traditional HPC workflows focus on parallel processing for large-scale simulations and data-intensive tasks, while modern AI/ML workflows emphasize iterative model training, data preprocessing, and optimization. They often leverage specialized hardware (e.g., GPUs) to perform tasks like deep learning and pattern recognition on large datasets. They consist of multiple complex tasks or stages that continue to communicate and exchange data with each other. Tal *et al.* [3] highlight some key differences between traditional and modern HPC AI/ML workflows:

- **Distributed systems.** They require multiple nodes on heterogeneous resources that span CPUs, GPUs, and FPGAs. This makes network and Storage I/O become much more important.
- **Orchestration.** Workflows require scheduling different types of computations on heterogeneous infrastructures [18], [36], which can be unpredictable. We may need to dynamically initiate ML or simulation jobs for training or inference. Additionally, workloads can increase or decrease over time, which MPI ranks are not well-equipped to handle.
- **Complex scheduling.** Large number of jobs may be required to run on CPUs and GPUs for training and uncertainty quantification. Existing HPC schedulers like Slurm [37] cannot co-schedule heterogeneous resources and do not scale well to millions of training jobs that may be required on simulation data.
- **Monitoring.** They may require a coarse level of monitoring along the critical path, given their complex nature

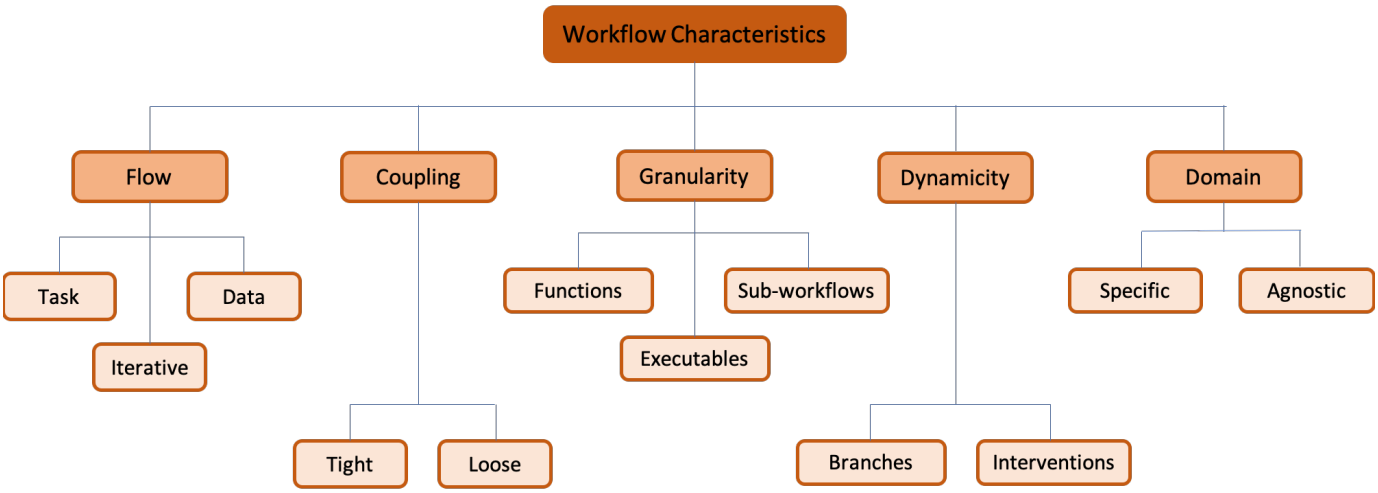


Fig. 4. One of the five axes, Workflow Characteristics, with corresponding terms and sub-terms that categorize workflows running on workflow management systems (adapted from [35]).

and codebases. Each component may require unique monitoring and logging mechanisms, and their evaluation criteria may differ.

- **Robustness.** State management in modern workflows may be significantly different than traditional checkpointed simulations. This means that fault tolerance strategies can vary throughout the workflow.

C. Workflow Management Systems (WMS)

A Workflow Management System (WMS) (or sometimes Scientific Workflow Management System, SWfMS) is a software that allows scientists to define, compose, and execute computational and data processing pipelines made of interdependent tasks. Tasks may involve simulations, data transformation, analysis, visualization, etc., arranged into dependency graphs (often DAGs), with edges representing either data flow or control dependencies. The WMS handles orchestration (scheduling, dispatching tasks), data movement, fault tolerance, provenance (recording how outputs were derived), resource allocation, and often supports heterogeneous compute/storage infrastructures [38].

Ferreira da Silva *et al.* classifies WMSs in their work according to the execution model, the handling of heterogeneous environments, and the data access methods [2]. However, more recent work by Suter *et al.* codifies a set of axes to compare and describe different WMSs. Figure 4 shows one of the five axes, workflow characteristics, which is the most relevant to this study. [35]. However, following are the brief descriptions of each of these axes:

- **Workflow Characteristics.** This axis captures how workflows are organized (tasks vs data-driven), how complex the components are, what kinds of dependencies connect components, and whether execution paths can change at runtime. These structural features strongly shape how a workflow system can optimize scheduling, resource use, and performance adaptivity.
- **Composition.** This axes refers to how workflows are defined, configured, and organized in a WMS. Overall, this axes helps users with different technical backgrounds understand how accessible WMSs are.
- **Orchestration** describes how a WMS implements and manages execution of workflow tasks. Whether task launching and scheduling are static or dynamic, whether triggers/events at runtime can influence execution, and how resources are allocated (directly by the WMS, delegated to external resource managers, or cloud-based). These decisions affect how efficiently workflows use computing infrastructure.
- **Data Management** describes how workflows handle data through their lifecycle: how data moves between tasks, where and when data is stored (intermediate, final outputs), and how access patterns are optimized (caching, layout, transfer strategies). These strategies are especially crucial in data-intensive workflows and heavily influence performance.
- **Metadata Capture** involves gathering contextual data during workflow execution—such as provenance (both what the workflow is supposed to do and what it actually did), runtime state, performance monitoring, and detecting anomalies. These capabilities are essential for reproducibility, debugging, and optimization.

Table I summarizes the types of workflows each WMS is able to support and execute based on the sub-terms in the first axis (Workflow Characteristics). Workflow Characteristics is more focused on the types of workflow which a WMS can support than on the characteristics of the WMS itself, i.e., on *what* the workflow does. Hence, the first axis is the most relevant to this survey and we explain its corresponding and sub-terms next, which are used in Table I. First, **flow** is one of the important workflow features which directly impacts its optimization by WMSs. Workflows have different components which get inputs, do processing, generate outputs, and then

TABLE I
CLASSIFICATION OF SOME SCIENTIFIC WORKFLOW MANAGEMENT SYSTEMS BASED ON WORKFLOW CHARACTERISTICS [35].

WMS	Flow	Granularity	Coupling	Dynamicity	Domain
AiiDA	Task, Iterative	Sub-workflows, Executables, Functions	Loose	Branches, Intervention	Agnostic
COMPSs	Task, Iterative	Sub-workflows, Executables, Functions	Loose	Branches	Agnostic
Cylc	Task, Iterative	Sub-workflows, Executables	Loose	Branches, Intervention	Agnostic
FireWorks	Task	Sub-workflows	Tight	Branches	Agnostic
Galaxy	Data	Sub-workflows, Executables	Loose	Branches, Intervention	Agnostic
Merlin	Task, Iterative	Sub-workflows	Loose	-	Agnostic
Nextflow	Data	Sub-workflows	Loose	Branches	Agnostic
Parsl	Data	Sub-workflows	Loose	Branches	Agnostic
Pegasus	Data	Sub-workflows, Executables	Loose	Branches	Agnostic
Radical	Task, Iterative	Functions	Tight	-	Agnostic
Snakemake	Task, Iterative	Sub-workflows, Executables, Functions	Loose, Tight	Branches	Agnostic
Swift/T	Task, Data	Functions	Tight	Branches	Agnostic
Kepler	Data	Sub-workflows, Executables, Functions	Loose	Branches, Intervention	Agnostic

terminate. The structure of a workflow is defined by all these **tasks** performed by the workflow components. All these different tasks in a workflow can be executed multiple times in an **iterative** manner. The **data** that flows within the workflow components may also drive the structure and execution of the overall workflow.

Second, **coupling** of workflow tasks is another prominent workflow feature. Coupling refers to the degree of interdependence between workflow tasks. Tasks with **tight** coupling are heavily dependent on each other's internal details (data, order, or state), while loosely coupled tasks interact through well-defined interfaces and can be changed or executed independently.

Third, the **granularity** of individual workflow tasks also causes differences in workflow structures. Some workflows may only have multiple **function** calls to get complex tasks done, while others may exist of a script of programs. However, the most common definition of a workflow is the composition of **standalone executables**, each of which consists of multiple function calls with inputs and generated outputs to complete complex tasks. As the complexity of workflows increases, it is not uncommon to express them in a hierarchical format consisting of **sub-workflows**.

Fourth, the **dynamicity** of a workflow is its ability to modify its structure during execution based on **conditional branches**. These branches are activated when a pre-defined condition is realized or due to an external event. Another way to activate these branches is through **runtime interventions** where control is given back to the user or two an automated external decision process. Lastly, different workflow **domains** are served by different WMSs. Some are domain-**specific** while others are application-**agnostic** [35].

1) WMSs for AI/ML-enabled Scientific Workflows: With the growth in complexity and introduction of AI/ML in scientific workflows, WMS designers need a broader and deeper understanding of workflow requirements and behaviors to have improved algorithms for resource provisioning, computational job scheduling, and data management. First, in order for WMSs to support large scale, ML workflows, a shared NFS file system should be a top feature. This will make the system much more convenient as compared to when file accesses are done over the network. Second, scientific WMS will need to adapt to dynamic changes on the number of input and output files in order to support ML workflows. This means that input and output files for each job will not be known before the workflow's execution begins [30].

Following is a list of few WMS which support some level of data collection, setting the stage for performance analysis.

a) Pegasus: The Pegasus WMS enables scientists to specify complex computational pipelines as high-level workflow descriptions, which are then mapped to diverse execution environments such as campus clusters, national cyber infrastructure, and cloud platforms [39]. By abstracting workflow specification from resource details, Pegasus supports portability and scalability of scientific workflows across heterogeneous systems. In addition to core orchestration capabilities, Pegasus has been extended to support performance analysis using tools like `pegasus-statistics` for performance metrics, `pegasus-analyzer` for debugging, `pegasus-plots` for visualizations, and `pegasus-monitord` for runtime monitoring, capturing job execution times, data transfer overhead, and resource utilization [40]–[42]. This monitoring capability allows users to analyze end-to-end workflow performance, track job and data dependencies, and derive insights about bottlenecks, making Pegasus not only a workflow engine but also a platform that supports performance-aware execution.

b) Nextflow: Nextflow [43] is a WMS that provides built-in performance monitoring through execution reports, trace files, and timeline visualizations that capture CPU usage, memory consumption, task duration, and I/O operations for each workflow process. The system generates HTML execution reports using the `-with-report` option and creates trace files with the `-with-trace` option containing submission time, completion time, and resource usage statistics [44]. For real-time monitoring, Nextflow integrates with Tower (now Seqera Platform), which tracks workflow execution across any infrastructure and provides resource optimization recommendations based on actual usage patterns. These performance analysis capabilities enable scientists to identify bottlenecks, optimize resource allocation, and improve computational effi-

ciency across distributed computing environments.

c) Snakemake: Snakemake [45] is a workflow management system designed to create reproducible and scalable data analyses using a Python-based domain-specific language. Workflows are defined through rules that specify how to generate output files from input files, with dependencies automatically determined to create a directed acyclic graph of jobs that can be parallelized. For performance analysis, Snakemake provides built-in benchmarking capabilities that allow rules to capture execution metrics including runtime, memory usage, and I/O operations [46]. These benchmark results are stored in tab-separated files that can be used to build complex performance analysis pipelines.

d) Kepler: Kepler [47] is an open-source scientific workflow system that provides a graphical user interface for designing, executing, and sharing workflows using an actor-oriented modeling approach. Workflows are composed of modular components called actors connected through channels that represent data flow. For performance analysis, Kepler provides a comprehensive provenance framework that automatically captures process and data monitoring information, including workflow execution context, parameter settings, resource usage, and data lineage across distributed computing environments [48]. The provenance collection mechanism uses an event-based architecture designed for minimal performance overhead, recording actor executions, data transformations, and workflow evolution to enable reproducibility. These performance analysis capabilities enable scientists to validate experimental procedures.

Key Takeaways

Scientific workflows have become integral to managing the complexity of modern compute- and data-intensive scientific tasks. Represented as Directed Acyclic Graphs (DAGs), workflows automate and orchestrate interdependent tasks to enhance scalability, reproducibility, and provenance. Workflow Management Systems (WMSs) such as Pegasus, Nextflow, and Snakemake enable seamless execution, data movement, and performance monitoring across heterogeneous infrastructures. The integration of AI/ML into workflows, through modes such as ML-in-, ML-out-, and ML-about-HPC, has further expanded their capabilities, enabling adaptive, data-driven scientific discovery. These AI/ML-enabled workflows introduce new challenges in scheduling, orchestration, and monitoring, requiring WMSs to evolve with advanced support for dynamic data dependencies, heterogeneous resources, and performance-aware execution.

IV. Performance Analysis of HPC Applications

Over the years, various performance analysis tools have been developed to meet the diverse needs of HPC users. Figure 5 shows a classification of HPC performance analysis tools. These tools generally fall into two categories: (1) *sampling*-based tools, which collect data periodically or on specific events, and (2) *instrumentation*-based tools, which require adding measurement markers to the source code. The data formats produced by these tools are typically categorized into traces and profiles. Sampling-based tools almost always produce profiles. Tools such as Vampir [49], Paraver [50], and Scalasca [51], record traces with detailed sequences of events during program execution, useful for in-depth temporal analysis. Hybrid tools combine sampling, instrumentation, and tracing to provide comprehensive performance insights. System-level monitoring tools, such as PAPI [52], [53] and LDMS [54], is a separate category of performance analysis tools that monitor overall system performance metrics, aiding in identifying system-wide bottlenecks. These tools are often incorporated as part of hybrid tools. Table II shows the different characteristics supported by performance analysis tools in a binary format.

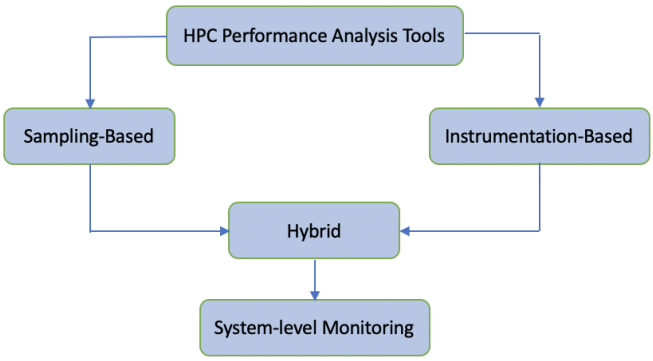


Fig. 5. Classification of HPC Performance Analysis Tools.

Sampling tools (e.g. HPCToolkit [55], Open|SpeedShop [56], `mpiprof`) are useful when the source code is too large or not accessible, offering profiling without code changes. However, they gather data across the entire application, which can overwhelm analysis and obscure critical regions. Their fixed sampling rate also poses challenges: too high increases overhead and I/O issues, while too low risks missing important behavior. Since HPC applications often have distinct phases, a uniform sampling rate is suboptimal.

In contrast, instrumentation-based tools such as TAU [57], Score-P [58], Timemory [59], and Caliper [60], [61] offer more control and lower overhead by targeting specific code regions. This makes them well-suited for analyzing phase behavior and critical paths. However, they require access to and familiarity with the application source code. In addition, common runtime characteristics collected by performance tools include function-level timings, call counts, processor usage, and memory usage. Traces are often visualized and capture fine-grained event timelines for each process/thread, enabling detailed analysis of communication delays, load imbalance, synchronization overhead, etc. Finally, some frameworks support continuous or online monitoring but, they typically focus on system metrics rather than application-level behavior. The contextual features of the performance analysis

tools mentioned in the characterization in Figure 5 can be found in Table III.

A. Performance Analysis for Bulk-Parallel HPC Applications

In bulk-parallel HPC applications, performance analysis tools need to meet several core requirements. They must support instrumentation across tightly coupled compute, communication, memory, storage, and I/O subsystems, capturing metrics such as per-rank execution time, message volumes, and hardware counter data. Because these applications run on extreme scale, the measurement overhead must remain minimal and the data collection must be manageable. Effective tools thus provide both profiling (aggregated metrics) and tracing (time-stamped event logs) so that bottlenecks can be uncovered either through high-level summaries or detailed temporal analysis.

The performance analysis of Bulk-Parallel HPC applications usually involves three steps: source code instrumentation or sampling, performance data collection, and performance data analysis and visualization. There are a number of tools that provide one or more of these capabilities. This section briefly describes the functionality and support provided by state-of-the-art performance tools.

1) TAU Performance Suite: The TAU Performance System [57], [63] is a comprehensive performance analysis toolkit that supports various programming paradigms, including MPI, OpenMP, and hybrid models. It is compatible with languages such as C, C++, Fortran, Java, and Python. TAU's architecture comprises three primary components: instrumentation, measurement, and analysis.

TAU collects performance data through profiling and tracing. Source code instrumentation can be done either manually or automatically, and can be applied at various levels, including functions, methods, basic blocks, and statements. Additionally, TAU integrates with hardware performance counters (e.g., via PAPI), enabling access to low-level metrics.

Recent developments have focused on preparing TAU for exascale computing environments by improving scalability, support for heterogeneous architectures, and integrating with emerging programming models [64]. For workflow performance analysis, TAU improved instrumentation methods to capture data across multiple workflow components [65]. The profiles and/or traces collected by each component are aggregated with post-processing scripts that generate structured JSON output and merged traces.

2) HPCToolKit: HPCToolkit [55], [66] is an integrated suite of tools designed for performance measurement and analysis of parallel applications. It supports a variety of programming models, including MPI, OpenMP, and CUDA, and is compatible with languages such as C, C++, and Fortran. HPCToolkit's architecture comprises three primary components: measurement, analysis, and visualization.

HPCToolkit collects performance data by sampling and tracing. It employs asynchronous sampling to gather call path profiles, capturing metrics such as CPU cycles and cache misses. Further, HPCToolkit's analysis tools attribute performance metrics to calling contexts, enabling developers to pinpoint where in the program bottlenecks occur and identify their underlying causes. It supports fine-grained call path tracing, allowing for detailed examination of application behavior over time. `Hpcviewer` and `hpctraceviewer` are HPCToolkit's visualization tools. These tools offer graphical interfaces for exploring performance data, presenting call path profiles and traces in a hierarchical manner.

HPCToolkit's capabilities have been extended to support performance analysis of GPU-accelerated applications, addressing the challenges of performance analysis on emerging exascale supercomputers [67]. Moreover, it is important to note that while HPCToolkit excels at analyzing individual parallel application components, it is not inherently designed for end-to-end analysis of complex scientific workflows composed of heterogeneous tasks or mixed-language components. In such cases, especially where workflows involve diverse execution models, complex task dependencies, and data flows, additional integration effort may be required. For comprehensive workflow-level performance insights, HPCToolkit is best used in conjunction with tools specifically tailored to scientific workflow analysis.

3) Caliper: Caliper [61] is a lightweight, modular performance analysis library designed to integrate performance measurement capabilities directly into high-performance computing (HPC) applications. Developed at Lawrence Livermore National Laboratory, Caliper supports C, C++, and Fortran programs, and is compatible with parallel programming models such as MPI, OpenMP, CUDA, and Kokkos.

The framework enables developers to annotate the source code with performance markers, facilitating the collection of rich contextual information during program execution. Caliper's architecture allows for flexible configuration at runtime, supporting various measurement techniques including sampling, tracing, and hardware counter integration. The collected data can be exported in multiple formats, allowing seamless integration with analysis tools such a Hatchet and visualization platforms such as Chrome's trace viewer [60].

Caliper's design emphasizes minimal overhead and ease of integration, making it suitable for both large-scale HPC applications and smaller-scale performance studies. Its extensible plugin system provides flexibility and customization by allowing developers to enable or disable various services at runtime.

In terms of scientific workflow performance analysis, Caliper may not provide holistic workflow-level performance insights out-of-the-box. This is particularly true in the case of multi-cluster and multi-binary workflows. Although Caliper excels at fine-grained performance analysis of individual components within a workflow, integrating Caliper with other tools or frameworks specifically designed for workflow-level performance analysis could offer a more comprehensive understanding of the workflow's performance characteristics. Especially in scenarios involving complex task dependencies and diverse execution models.

TABLE II
CHARACTERISTICS OF SOME OF THE PERFORMANCE PROFILING, TRACING, AND MONITORING TOOLS

Tool	Methods		Outputs		Monitor	CPU	GPU	Rank	Thread	Perf Counters	Workflow Aware
	Instrumentation	Sampling	Trace	Profile							
TAU	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
HPCToolkit	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Caliper	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Score-P	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Open|SpeedShop	✗	✓	✓	✓	✗	✓	✗	✓	✓	✓	✗
MPIProf	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
timemory	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Scalasca	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Darshan	✗	✓	✓	✓	✗	✓	-	✓	✗	✓	✓
Intel Advisor	✓	✓	✗	✓	✗	✓	✓	✗	✓	✓	✗
Recorder	✗	✓	✓	✗	✗	✓	-	✓	✗	✓	✓
NVIDIA Nsight	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
PerfFlowAspect	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓
Chimbuko	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓
SOMA [62]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DFTracer	✓	✗	✓	✗	✗	✓	✗	✓	✓	✗	✓

TABLE III
CONTEXTUAL FEATURES OF SOME OF THE PERFORMANCE ANALYSIS TOOLS

Tool	Programming Models	Languages	Scalability	Overhead	Visualization / Output
TAU	MPI, OpenMP, CUDA, HIP, Pthreads, OpenCL	C, C++, Fortran, Python, JAVA, etc.	High	Medium	PerfExplorer, Vampir, Jumpshot ParaProf, Paraver, OTF2
HPCToolkit	MPI, OpenMP, CUDA, HIP, Pthreads	C, C++, Fortran, Python	High	Low	hpcviewer, hpctraceviewer
Caliper	MPI, OpenMP, HIP, CUDA	C, C++, Fortran, Python	High	Low	Hatchet, JSON, CSV exports
Score-P	MPI, OpenMP, Pthreads, CUDA, OpenCL	C, C++, Fortran, Python	Medium	Medium	Vampir, Scalasca, Cube4, OTF2
Open|SpeedShop	MPI, Pthreads, OpenMP	C, C++, Fortran	Medium	Medium	OSS GUI, CLI
MPIProf	MPI	C, Fortran	Very High	Very Low	Text/CSV logs
timemory	MPI, OpenMP, CUDA, HIP	C, C++, Fortran, Python	Very High	Low	Hatchet, Perfetto, JSON, XML
Scalasca	MPI, OpenMP, Pthreads	C, C++, Fortran	High	Medium	Scalasca GUI, Cube4, OTF2
Darshan	MPI-I/O, POSIX I/O, HDF5	C, C++, Fortran	Very High	Very Low	PyDarshan (PDF reports), Darshan logs
Intel Advisor	OpenMP, MPI, OpenCL, SYCL	C, C++, Fortran, Python	High	Medium	Intel Advisor GUI, Roofline
Recorder	MPI-I/O, POSIX I/O, HDF5	C, C++, Fortran	High	Low	I/O traces, Plots/Parsers
NVIDIA Nsight	MPI, OpenMP, CUDA, OpenACC	C, C++, Fortran	High	Medium	Nsight GUI, Roofline
PerfFlowAspect	MPI, OpenMP, CUDA, HIP	C, C++, Python	High	Low	Perfetto, Chrome trace
DFTracer	POSIX I/O	C, C++, Python	High	Low	Perfetto, DFAnalyzer

4) Score-P: Score-P [58] is a scalable and modular performance measurement infrastructure developed collaboratively. It provides a unified instrumentation and measurement system for profiling and tracing parallel applications, supporting various programming models such as MPI, OpenMP, CUDA, OpenCL, and OpenACC. Since standard Python performance analysis tools cannot cope with highly parallel programs, Andreas *et al.* demonstrate advanced Python Performance Monitoring with Score-P [68].

The framework enables developers to collect performance data through automatic instrumentation, manual annotations, or sampling methods. The collected data can be stored in standardized formats like OTF2 [69] for traces and CUBE4 for profiles, facilitating interoperability with analysis tools such as Vampir and Scalasca. Lastly, Score-P's flexible configuration allows users to control measurement granularity and overhead through environment variables, enabling tailored performance analyses.

Although Score-P excels at detailed performance analysis of individual components within HPC applications, it may not inherently provide holistic insights into complex scientific workflows composed of multiple heterogeneous tasks. In such scenarios, integrating Score-P with workflow-aware performance analysis tools or frameworks could offer a more comprehensive understanding of the workflow's performance characteristics.

5) Open|SpeedShop: Open|SpeedShop (O|SS) [56] is an open-source performance analysis framework designed to support a wide range of profiling and tracing experiments for parallel applications. It provides a unified interface for defining

"experiments," which encapsulate what performance metrics to collect and how to present them. O|SS supports program counter (PC) sampling, call-path profiling, hardware counter metrics, MPI profiling/tracing, I/O tracing, and floating-point exception detection. One of the core philosophies of O|SS is to apply instrumentation and monitoring to unmodified application binaries whenever possible [70].

O|SS is built atop components such as Dyninst/DPCL for binary instrumentation, libmonitor for intercepting I/O and system calls, PAPI for hardware counter access, and MRNet for data aggregation in parallel settings. In the O|SS workflow, a user selects an "experiment" (e.g. pcsamp, usertime, hwc, io, mpi, etc.), which drives specific collectors (sampling, tracing, or counter-based) and result views. Moreover, it supports multiple user interfaces (GUI, command line, Python API) and provides a plugin infrastructure so new measurement or analysis modules can be added modularly.

While O|SS offers extensive support for instrumentation, sampling, and tracing that could probe individual components or modules of scientific workflows, it is not inherently designed to treat a workflow as a holistic unit. Adapting O|SS for full scientific workflow analysis would require custom orchestration of experiments across all workflow components, synchronized trace merging, and a higher-level abstraction to relate performance data across task dependencies. Thus, for end-to-end workflow performance evaluation, combining O|SS with workflow-level performance tools is likely necessary.

6) MPIProf: MPIProf [71] is a lightweight, profile-based tool developed by NASA HECC for MPI applications, using the PMPI interface to collect aggregated statistics on MPI calls, I/O, and memory usage without requiring source code changes. It reports metrics such as number of calls, message sizes, and time spent in point-to-point and collective MPI routines, and optionally captures call-path context when compiled with debugging flags. MPIProf also supports profiling POSIX and MPI I/O operations, providing per-rank summaries of I/O behavior.

While MPIProf can help pinpoint MPI and I/O hotspots within components of scientific workflows, it is not inherently suited for full workflow-level performance analysis.

7) Timemory: Timemory [59] is a modular and extensible toolkit for performance measurement and analysis in HPC, designed to allow users to combine different performance "components" flexibly with minimal overhead. Although implemented in C++, timemory supports instrumentation in C, Fortran, Python, and CUDA, offering a unified interface across these languages. Users can create measurement bundles that mix timers, hardware counters, memory metrics, GPU counters, and more. These bundles can be enabled, disabled, or interleaved dynamically at runtime. One of timemory's design goals is that disabled instrumentation should impose near-zero cost, making it safe to integrate broadly without disturbing performance. The framework is built to interface with external tools and to be extended by users. If timemory does not natively support a measurement type, the user can plug in custom components.

Timemory is well suited to profiling and analyzing individual tasks or modules within scientific workflows, especially when these modules are implemented in languages and paradigms that timemory supports. Its flexibility in combining measurement types allows detailed characterization of computation, memory, and hardware counter behavior. However, timemory does not inherently provide workflow-level orchestration, merging of performance traces across multiple distinct binaries or languages, or correlation of performance data with task dependencies or data flow in workflows. To perform holistic workflow-level performance analysis, timemory would likely need to be paired with workflow-aware tools that can aggregate, align, and interpret performance data across task boundaries.

8) Scalasca: Scalasca [51] is a performance analysis tool developed for optimizing parallel applications that use MPI and/or OpenMP. It offers two primary modes: profiling and event tracing. In profiling mode, Scalasca generates aggregate call-path metrics and hardware counter data to identify hotspots. In tracing mode, it records detailed time-stamped events to detect wait states and communication/synchronization inefficiencies. The reports can be explored via the CUBE graphical browser or external timeline tools such as Vampir.

Scalasca can be applied to scientific workflows insofar as those workflows consist of large-scale MPI/OpenMP tasks or kernels: it can profile individual components, find communication bottlenecks, synchronization inefficiencies, and load imbalances. However, Scalasca is not inherently workflow-aware.

9) Intel Advisor: Intel Advisor [72] is a performance design and analysis tool aimed at helping developers optimize vectorization, threading, memory usage, and accelerator offloading in applications written in C, C++, Fortran, SYCL, Python, and others. It offers multiple "perspectives" (e.g. vectorization, roofline, offload modeling) to guide where performance improvements are possible. Its key features is Roofline Analysis, which plots application performance (FLOP/s) against arithmetic intensity to help identify whether kernels are compute-bound or memory-bound. Intel Advisor also supports offload modeling, which helps estimate which parts of a CPU code might benefit from being offloaded to a GPU and predicts the performance gains and bottlenecks.

In the context of scientific workflows, Intel Advisor is useful for profiling and optimizing individual computational kernels or components, especially those involving vectorization, threading, or offload to accelerators.

10) NVIDIA Nsight: Nsight Systems [73] is a system-wide performance analysis tool from NVIDIA, designed to capture CPU, GPU, OS, library, and interconnect activity across a unified timeline. It helps developers visualize dependencies, identify bottlenecks (e.g., between CPU and GPU work), monitor hardware utilization (SM occupancy, memory bandwidth, PCIe/NVLink throughput), and trace CUDA API calls. Nsight Compute complements this by being a detailed kernel-profiler: it provides per-kernel metrics, instruction throughput, memory behavior, occupancy, and recently includes Roofline analysis to

help locate whether kernels are compute-bounded or memory-bandwidth bounded.

Nsight is quite strong in analyzing individual tasks or kernels within scientific workflows, especially when those tasks make use of CUDA, GPU offload, or when CPU-GPU interactions are significant. Because it supports multi-node profiling in Nsight Systems, it can also scale to cluster settings to some degree. However, Nsight does not inherently capture workflow-level metadata such as task dependencies, data flow between workflow stages, or cross-language orchestration.

B. Performance Analysis for Complex Scientific Workflows

The complex nature of modern scientific workflows with AI/ML-assisted science requires different features in the performance tools for their effective performance analysis. In this section, we discuss some of them. First, considering that workflows consisting of multiple programs have multiple source code components, the tool should preserve the readability of the complex code base, should be minimally intrusive in terms of the amount of instrumentation code, and should also minimize introducing any instrumentation-related errors. It should be user-friendly without a steep learning curve and preferably have a human-readable trace file format that allows users to easily understand and validate the data. Second, given the emergence of complex AI/ML workflows that rely on both C/C++ and Python interactions, the tool should be able to support instrumentation and measurement across multiple languages with ease. Third, a tool that profiles scientific workflows effectively needs to support disparate data sources and generate trace files that are easily composable across different components (multiple binaries) and across heterogeneous systems (multiple HPC clusters). Fourth, given the sheer amount of data that a workflow can generate, the tool should be able to provide both coarse-grained, high-level analysis as well as finer-grained analysis with hardware performance counters if and when desired. Finally, the tool should be able to generate trace files that can be easily visualized with existing and commonly available visualization tools [75].

This section presents a discussion of some projects, techniques, and tools from the literature for the analysis and optimization of scientific workflows. Complementing this perspective, a recent survey on I/O analysis and tooling [76] underscores that AI/ML integration is making application behavior increasingly complex, which also strengthens the need for detailed I/O tracking and profiling.

1) Workflow Performance Profiles: Król *et al.* presents a holistic process for the development and analysis of Workflow Performance Profiles. A performance profile for a given job and a metric can be described as a time series with values of the metric measured in equidistant points in time during the job execution [77]. When data is collected from multiple executions of the same workflow configuration, statistically significant performance profiles for each job in a workflow can be computed. In order to generate workflow performance profiles, a workflow and a set of distinct input parameters are taken as inputs. For each possible combination of these parameters, a workflow execution is performed. The workflow performance data collected using the monitoring tools is then used to build average profiles.

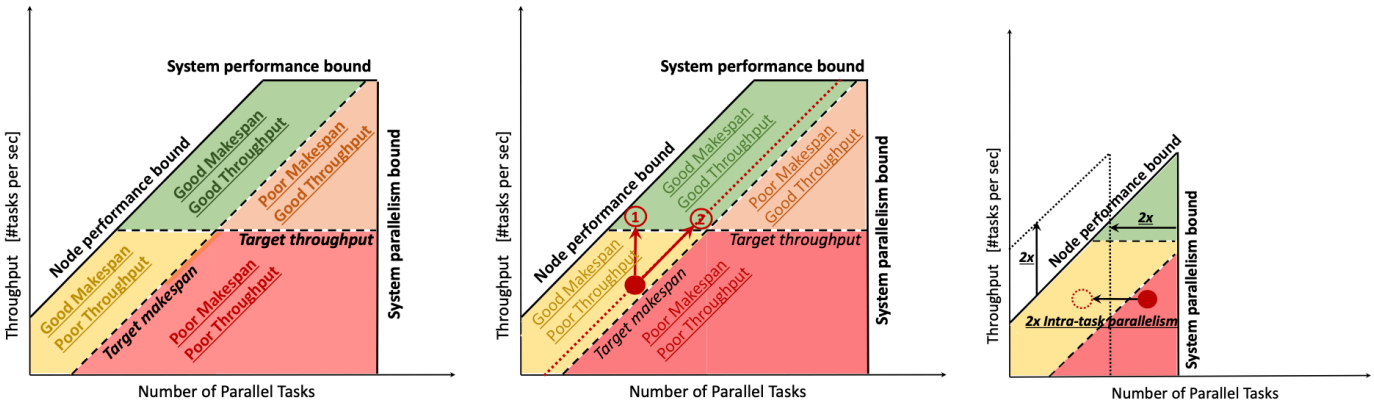
2) Workflow Roofline Model: In their work, Ding *et al.* introduce a Workflow Roofline Model, which ties a workflow's end-to-end performance with peak node- and system-performance constraints [74]. Using this new workflow Roofline model, both node performance (FLOPS, data movement in CPU/GPU, etc.) and system performance can be analyzed, which is an expansion of the original Roofline model to the workflow domain. It also allows for the analysis of potential performance bottlenecks (node-bound vs. system-bound), thereby guiding workflow performance optimizations. Figure 6 shows the Workflow Roofline interpretation for time- and throughput-sensitive workflows.

3) Top-down Performance Analysis: Tschueter *et al.* presents a top-down performance analysis methodology for scientific workflows by presenting event logs for both individual job steps and derived statistics for the workflow levels [78]. This work tries to widen the scope of performance analysis beyond computational performance by including queue waiting times, storage capacities, and data transfer speeds. The analysis starts with an overview of the entire workflow and then proceeds down the hierarchical structure of workloads, thereby increasing analysis detail. First, aggregated results are provided for the workflow level through visualization, which allows users to pinpoint problematic jobs. Once a job is selected, detailed information about the job is provided.

4) Stampede: Complex scientific workflows often experience failures due to temporary or localized resources problems. For example, a compute node may have a bad disk, or a network file server may be overloaded and time out. The Stampede (Synthesized Tools for Archiving, Monitoring Performance, and Enhanced DEbugging) project applies offline workflow log analysis to address reliability and performance problems for large, complex scientific workflows [79]. Specifically, Stampede integrates `NetLogger` [80] and `Pegasus-WMS` into a general-purpose framework to analyze the performance and failure states of running workflows. In general, Stampede has algorithms for workflow failure prediction, probabilistic anomaly detection, and a web-based dashboard to visually analyze and monitor workflows.

5) RAMSES Project: The Department of Energy RAMSES (Robust Analytic Models for Science at Extreme Scales) project, launched by the U.S. DOE in 2014, seeks to develop end-to-end analytical performance models for complex scientific workflows in extreme-scale environments. The project goals include building tools that can explain, predict, and optimize workflow performance by combining first-principles modeling (networks, storage, and compute) with data-driven calibration and models synthesis. A performance advisor tool and a database of observed behavior and parameter estimates are also part of its infrastructure.

6) PerfFlowAspect: PerfFlowAspect is a lightweight, open-source performance analysis tool specifically designed to



(a) Integrate target makespan and throughput

(b) Motivate two optimization directions

(c) Tell infeasible optimization directions

Fig. 6. The Workflow Roofline interpretation for time- and throughput-sensitive workflows [74].

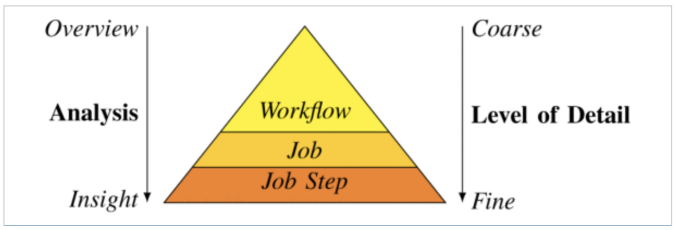


Fig. 7. The top-down workflow analysis methodology follows the workflows' hierarchy. The top level provides a general overview of the entire workflow while subsequent levels provide more detailed information for the individual workflow components [78].

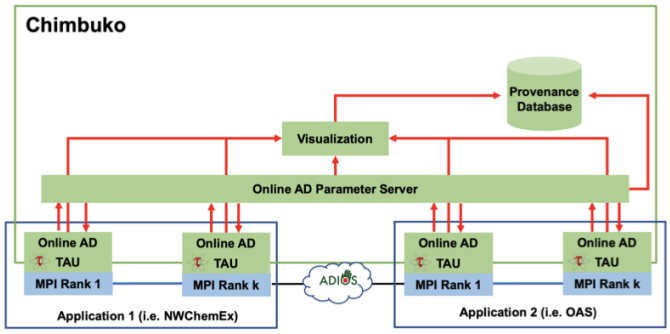


Fig. 8. Chimbuko's architectural diagram with its four major components. This image shows two applications running concurrently [85].

address the challenges posed by complex scientific workflows [75], [81], [82]. Unlike traditional performance tools that primarily target single-binary, bulk-synchronous applications, PerfFlowAspect is tailored for modern, heterogeneous workflows composed of multiple binaries, programming languages, and clusters.

```
1  import PerfFlowAspect
2  import PerfFlowAspect.aspect
3
4  @PerfFlowAspect.aspect.critical_path("around")
5  def foo(msg):
6      do_work_here()
7      return 1 if msg == "hello" else 0
8
9  def main():
10     foo("hello")
```

Listing 1. Python code with PerfFlowAspect annotated functions.

```
1  {"name": "foo", "cat": "__main__", "pid": 26356, "tid":
       26356, "ts": 1712881071887511.8, "ph": "X", "dur":
       3043.0},
```

Listing 2. PerfFlowAspect generated trace file with compact logging.

Built on aspect-oriented programming (AOP) principles, PerfFlowAspect introduces minimal, non-intrusive instrumentation, typically requiring only a single line of annotation per function (listing 1), thereby maintaining code readability and reducing user error. It supports C/C++, CUDA, and Python, and uses language-appropriate mechanisms (e.g., Python decorators and LLVM [83] passes for C++) to enable performance monitoring without burdening the user with steep learning curves or extensive code modifications. Moreover, PerfFlowAspect generates performance trace files (listing 2) in Chrome Tracing Format (CTF), making it composable and easily visualizable through tools like Perfetto. It also supports integration with Caliper, enabling the collection of low-level hardware counters without requiring additional annotations. This allows users to start with high-level, coarse-grained analysis and drill down into fine-grained metrics as needed.

PerfFlowAspect effectively captures detailed performance insights with minimal overhead (averaging 0.9%), supporting both holistic and application-level analysis. Its trace composability across binaries and systems, support for multiple languages, and flexible instrumentation make it particularly well-suited for emerging AI- and ML-driven scientific workflows operating in distributed HPC environments.

7) Chimbuko: Chimbuko [85] is a performance-analysis framework designed for large-scale HPC applications and scientific workflows, with a particular emphasis on online, distributed anomaly detection. Instead of collecting full traces, Chimbuko uses a combination of TAU instrumentation, provenance tracking, and an in situ analytics pipeline to iden-

TABLE IV
CHARACTERISTICS OF SOME OF THE VISUALIZATION TOOLS [84]

Tool	Trace		Profile		Architecture (Roofline)	Distributed Memory	Loop	Function	Workflow Aware
	Thread	Rank	Instrumentation	Sampling					
Intel Advisor	✗	✗	✗	✗	✓	✗	✓	✓	✗
Intel VTune	✓	✓	✓	✓	✗	✗	✗	✗	✗
NSight	✗	✗	✗	✗	✓	✗	✓	✓	✗
NSight Systems	✓	✓	✗	✓	✓	✗	✗	✗	✗
Paraprof	✓	✓	✓	✓	✗	✗	✗	✓	✗
Perfetto	✓	✓	✓	✓	✗	✗	✓	✓	✗
Hpcviewer	✓	✓	✗	✓	✗	✗	✓	✗	✗
Boxfish	✗	✗	✗	✗	✗	✓	✗	✓	✗
Vampir	✓	✓	✓	✓	✗	✓	✗	✓	✗
Jumpshot	✓	✓	✗	✗	✗	✓	✗	✗	✗
Grafana	✓	✓	✓	✗	✗	✗	✗	✗	✗
Hatchet	✗	✗	✓	✓	✗	✗	✗	✓	✗
Thicket	✗	✗	✓	✓	✗	✗	✗	✓	✗
Drishti	✓	✓	✗	✗	✗	✗	-	-	✗

tify anomalous events during execution and capture only their associated context. This significantly reduces trace data volume while enabling detailed, workflow-level diagnosis of performance issues such as load imbalance, unexpected communication behavior, or abnormal task runtimes. Figure 8 shows all the components in Chimbuko's architecture. Chimbuko supports interactive exploration through its web-based visualization interface and is designed to operate at extreme scales, making it suitable for modern workflow environments where applications are highly parallel, multi-component, and dynamically interacting. As such, it represents a shift toward performance-aware workflow monitoring that integrates provenance, anomaly detection, and trace analytics into a unified runtime system.

8) DFTracer: DFTracer [86] is a multi-level dataflow tracing tool designed to capture both application-level and low-level I/O events in complex, AI-driven scientific workflows. Traditional I/O profilers often lack the ability to correlate application behavior with system-level I/O operations, making it challenging to analyze workflows that integrate simulation, data analytics, and machine learning components. DFTracer addresses this gap by providing a unified tracing interface that captures events across different layers of the software stack.

The tool outputs trace files in Chrome Tracing Format (CTF), facilitating visualization and analysis using tools like Perfetto. DFTracer's optimized trace format and compression mechanisms enable efficient loading and analysis of large-scale workflows, with reported runtime overheads between 1-5% and trace sizes up to 7.1 times smaller compared to state-of-the-art tools. For analyzing the overlap between I/O and computation, DFTracer integrates with DFAnalyzer, which processes the generated traces to identify periods where I/O and computation are not effectively overlapped. This analysis helps to pinpoint I/O bottlenecks and optimize data loading strategies in workflows that utilize frameworks like PyTorch.

DFTracer has been demonstrated on real-world AI-driven workflows, such as MuMMI and Megatron-Deepspeed, showcasing its ability to capture multi-level performance data with minimal overhead. Its support for multiple programming languages, including C++, Python, and CUDA, along with its flexible instrumentation capabilities, make it a valuable tool for performance analysis in modern scientific workflows.

9) Recorder: Recorder [87] is a multi-level I/O tracing framework developed for HPC applications that captures I/O function calls from different layers of the stack (POSIX, MPI-IO, HDF5, NetCDF, PnetCDF) via function interposition, without requiring modifications or recompilation of the application. Users can control which I/O layers are traced, enabling flexible trade-offs between detail and overhead. One of Recorder's key innovations is its pattern-recognition-based compression algorithm, which identifies repeated I/O patterns both within individual processes and across multiple processes. This allows large scale runs to generate richly detailed traces while managing trace size growth and storage overhead.

Recorder is well suited for I/O-centric performance analysis in scientific workflows, especially when workflows involve heterogeneous I/O libraries (e.g., mixing HDF5, MPI-IO, POSIX) and layered tracing is required. It captures detailed I/O metadata (offsets, parameters, timing) that help in diagnosing I/O bottlenecks, overlapping I/O, and pattern dependencies across tasks. However, Recorder focuses on I/O tracing and does not by itself provide workflow task dependency metadata, nor automatic correlation across multiple binaries or languages in a workflow.

10) Darshan: Darshan [88] is a widely used HPC I/O characterization tool designed to capture the I/O behavior of applications with very low overhead. It intercepts I/O calls (POSIX, MPI-IO, etc.), records per-file/per-process statistics such as operation counts, access sizes, timing, and access patterns, and compresses/logs them for postmortem analysis.

Darshan is well suited for analyzing I/O behavior within scientific workflows, especially for understanding I/O bottlenecks in component binaries or for comparing I/O patterns across tasks. Because Darshan is lightweight, it can be enabled broadly without overly perturbing the application performance.

Key Takeaways

The HPC community maintains a diverse tool, each optimized for a specific balance of accuracy, overhead, scalability, and context of use. Tools differ in their design choices, such as the use of instrumentation or sampling. Instrumentation provides precise, event-level detail at higher cost, while sampling offers scalable, low-overhead insights with less precision. Similarly, profile- and trace-based tools cater to different analysis needs, which are aggregated summaries versus temporal behavior. These methodological and architectural choices reflect the target use case of each tool. An emerging use case is that of complex scientific workflows that have different requirements than bulk-parallel applications when it comes to performance analysis tools. These requirements include, but are not limited to, minimal intrusion, ease of use, multi-language support, data composibility, coarse- and fine-grained analysis, and visualization.

V. Visualization

Analyzing and interpreting performance data from parallel applications is inherently challenging. The situation becomes even more complex in scientific workflows, which generate vast and heterogeneous performance data across multiple components and execution stages. This complexity highlights the critical role of effective performance visualization techniques and tools to facilitate exploration, understanding, and optimization. Tools such as VisIt [89] have long focused on providing scientific visualization capabilities for extremely large datasets that have been generated on parallel clusters. However, the visualization requirements for complex scientific workflows are different. It is important to ingest and have holistic visualization capability for data coming from disparate sources in the form of DAGs, gantt charts, etc. This would allow users to analyze workflow data at a coarser and higher level before looking at task- or application-level data, if required.

Isaacs et al. organize a taxonomy of performance visualization in four *contexts*: hardware, software, tasks, and application [90]. Hardware covers the individual hardware elements as the performance data directly related to nodes, cores, etc. The software context covers the source code or the application being run. Tasks contexts encompass the individual tasks that are performed as part of the computation. Lastly, the application context refers to what is actually being computed, i.e., the mathematical details, etc. Some visualizations may be characterized under more than one of these contexts.

The contexts provided by Isaacs et al. help us to understand what kinds of performance visualization and tools already exist. The authors also highlight the challenges and opportunities for improvements in the performance visualization space. Table IV shows the different characteristics of the performance visualization tools in compact and binary format. In this section, we discuss the properties and capabilities of some of these visualization tools and techniques, especially in the context of complex scientific workflows. We begin by looking at a few WMS that provide workflow visualization support.

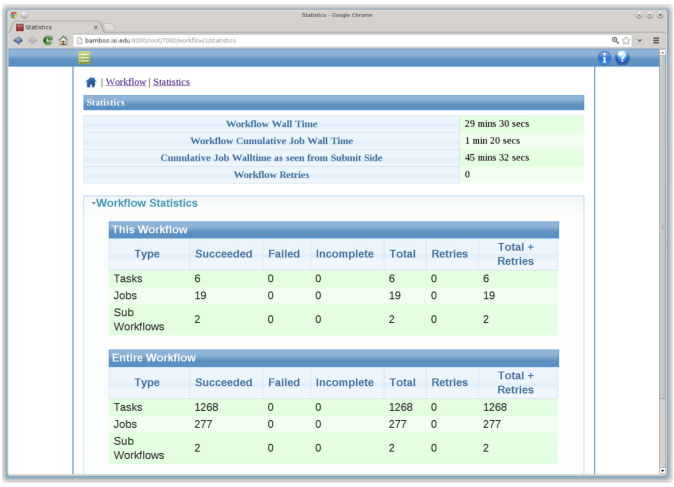


Fig. 9. Workflow Statistics displayed in `pegasus-dashboard` [39].

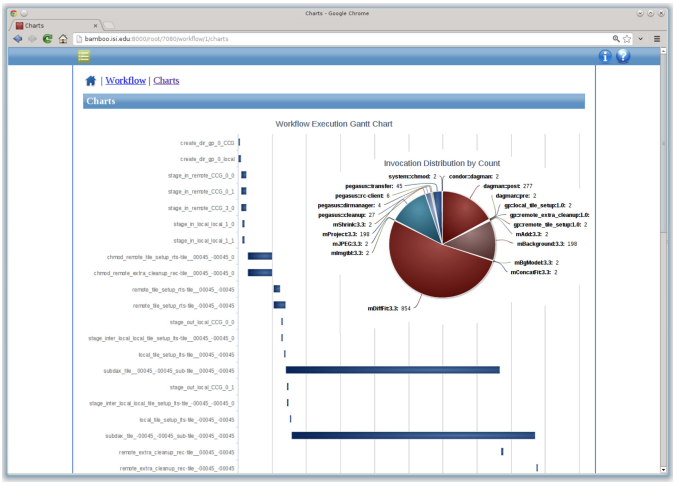


Fig. 10. Workflow Gantt Chart and Job Distribution Pie Charts displayed in `pegasus-dashboard` [39].

A. Pegasus WMS Visualizations

Pegasus WMS provides built-in visualization and monitoring capabilities that help users track the execution and performance of complex workflows across distributed resources [39], [91]. Its dashboard and monitoring tools visualize the directed acyclic graph (DAG) of the workflow, showing task dependencies, execution states, durations, and resource mappings, thereby offering a holistic view of workflow progress, as shown in figures 9 and 10. These visualizations are primarily designed for workflow-level provenance and performance monitoring rather than low-level performance profiling. Consequently, Pegasus effectively supports end-to-end workflow visualization and diagnosis but must be coupled with external profilers to capture detailed performance metrics at the application or kernel level.

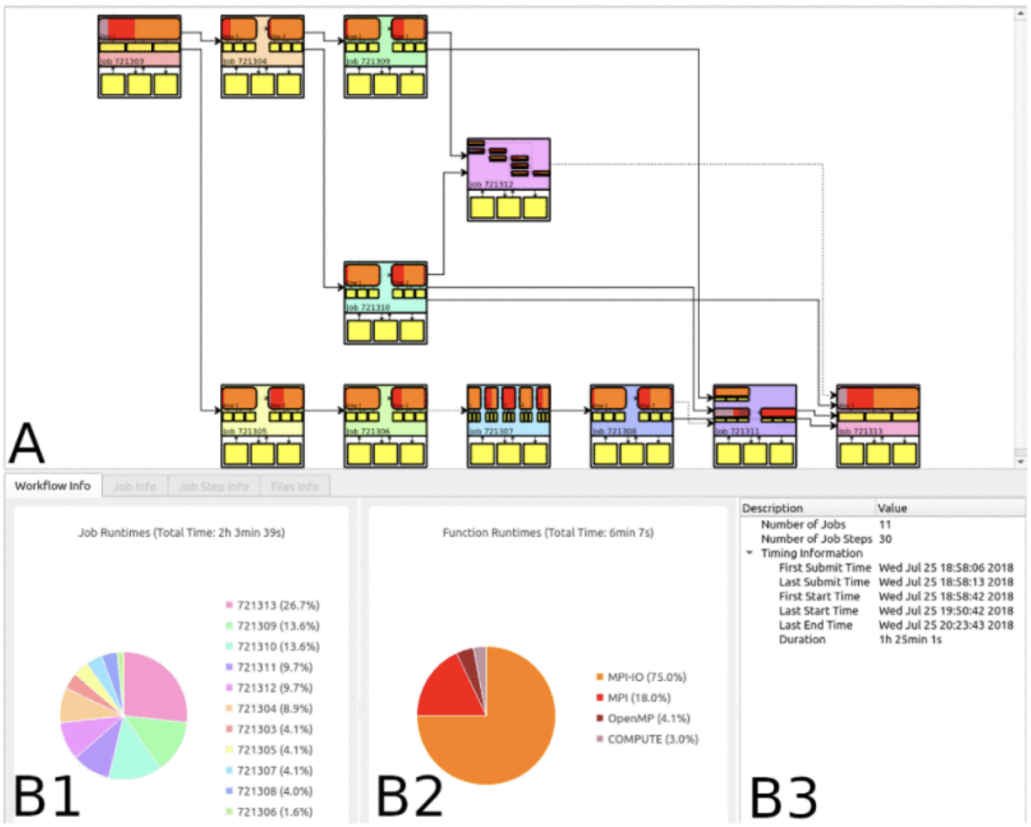


Fig. 11. Top-down performance analysis and visualization methodology: Workflow Visualizer main window consisting of Workflow Graph (A), Job Summary (B1), Function Runtimes (B2), and Info Table (B3) [78].

B. Top-down Visualization

The top-down performance analysis methodology (discussed in IV-B3) provides visualization for different data granularities. Figure 11 shows the top-level *Workflow View* window consisting of *Workflow Graph* at the top and *Info View* at the bottom. Users can choose to open a single job log file or a folder that contains all workflow job log and profile data files as they move *down* in their workflow analysis.

C. Intel Advisor

Intel Advisor (see IV-A9) is a performance analysis and optimization tool that helps identify vectorization, threading, memory, and GPU offload opportunities in C, C++, Fortran, and Python applications [72], [92]. Its roofline visualization provides an intuitive view of computational intensity and hardware utilization. However, Intel Advisor operates at the level of a single application or binary: the user instruments or runs the executable under Advisor, collects metrics, and then performs interactive visualization for each application run. Although it supports MPI and hybrid parallelism, it is not designed to integrate or visualize performance data from multiple, heterogeneous workflow components. In the context of complex scientific workflows comprising many interdependent steps, the tool may therefore be useful for analyzing individual program instances but less suitable for holistic, cross-step workflow performance visualization unless manual aggregation is applied.

D. Intel Vtune

Intel VTune Profiler is a comprehensive performance analysis tool that collects and visualizes detailed hardware and software performance data to identify bottlenecks in parallel applications [93]. It supports C, C++, Fortran, and Python programs, offering timeline and hierarchical views of CPU utilization, threading behavior, memory bandwidth, vectorization efficiency, and GPU offload performance. VTune's visualization environment enables developers to explore performance metrics interactively within a single application or binary. However, like Intel Advisor, VTune operates at the program level and does not natively integrate or visualize performance data from multiple, heterogeneous workflow components. As such, it is effective for deep per-application profiling, but less suitable for holistic, workflow-level performance visualization without external data aggregation.

E. Nsight Compute

NVIDIA Nsight Compute is a kernel-level performance analysis tool designed for CUDA applications running on NVIDIA GPUs [94], [95]. It provides detailed performance metrics, source-level correlation, and interactive visualization

of GPU kernel execution, including memory throughput, instruction mix, occupancy, and warp efficiency. Nsight Compute's GUI and command-line interfaces allow users to analyze bottlenecks at the kernel and instruction levels within a single GPU-enabled application. However, it is limited to profiling individual executables or kernels and does not natively support integrated visualization of performance data from multiple, heterogeneous workflow components. For multi-application scientific workflows, Nsight Compute is best suited for component-level GPU performance characterization rather than holistic workflow-level performance visualization.

F. Nsight Systems

NVIDIA Nsight Systems is a system-wide performance analysis tool that captures and visualizes the runtime behavior of heterogeneous applications across CPUs and GPUs [73], [96]. It provides timeline-based visualization showing interactions between processes, threads, CUDA kernels, memory transfers, and operating system activities, enabling users to identify bottlenecks and optimize concurrency between CPU and GPU workloads. Unlike Nsight Compute, which focuses on single-kernel analysis, Nsight Systems offers a broader execution view across multiple processes and libraries. However, while it can correlate activity across distributed or multi-process applications, it is primarily designed for profiling a single workflow run or binary ensemble rather than integrating performance data from diverse workflow systems. As such, it serves well for end-to-end tracing of GPU-accelerated workflow components but is not a native solution for holistic, workflow performance visualization.

G. Paraprof and PerfExplorer

TAU offers tools such as ParaProf and PerfExplorer for performance data analysis [63], [97], [98]. ParaProf provides a graphical interface for viewing performance profiles, including call-path trees, bar charts, and scatter plots, enabling users to explore metrics across threads, processes, and functions. PerfExplorer complements this with advanced data mining, statistical analysis, and comparative studies across multiple runs or components. When performance data from multiple workflow steps or applications is collected into TAUdb, these tools can aggregate and visualize the metrics together, providing a semi-holistic view of workflow performance. However, ParaProf and PerfExplorer are not inherently workflow-aware: they do not visualize DAG structure or task dependencies, and effective workflow-level analysis requires that all components be instrumented and their data imported into TAUdb. These tools are therefore best suited for analyzing aggregated performance metrics across heterogeneous workflow components, rather than providing structural workflow visualization.

H. Perfetto

Perfetto is an open-source performance analysis and visualization framework developed by Google for profiling and tracing complex software systems [99]. It provides a browser-based interface for interactive timeline visualization of events collected from CPUs, GPUs, and system processes, enabling users to explore correlations between tasks, threads, and I/O activities. Perfetto supports Chrome tracing format and can aggregate traces from multiple applications or processes, offering a global system view similar to Nsight Systems. Although originally designed for Android and Chrome performance analysis, it has evolved into a general-purpose tracing framework applicable to HPC and workflow environments where distributed tracing is available. However, Perfetto itself does not provide native workflow-level abstraction (e.g., DAG or step-level visualization); such integration must be achieved through custom trace merging or higher-level tooling.

I. Hpcviewer

HPCViewer is the interactive visualization component of the HPCToolkit performance analysis suite, providing a rich GUI to explore hierarchical performance measurements and calling context trees collected from parallel applications [55]. It enables users to navigate through calling contexts, correlate performance metrics such as time, hardware counters, and memory usage, and compare different program executions. Its top-down and bottom-up tree views, along with flame-graph-like visualizations, make it effective for identifying performance bottlenecks within a single application. However, HPCViewer operates primarily at the per-application or per-binary level; it visualizes data from individual HPCToolkit measurement databases rather than aggregating across multiple, heterogeneous workflow components. For workflow-oriented analysis, it could assist in profiling each step's performance but lacks inherent support for holistic, cross-component workflow visualization.

J. Boxfish

Boxfish is a domain-aware performance visualization platform developed by Lawrence Livermore National Laboratory that enables interactive exploration of performance data across multiple domains, for example, application communication patterns, hardware network topology, and system link traffic [100]. It supports filtering, querying, and projecting data from one domain into another allowing users to correlate communication, compute, and topology in a unified view. Although Boxfish is powerful for multi-domain correlation and supports large-scale system visualization, it is not explicitly designed for workflow-step abstraction; it assumes that the data pertains to one application's execution context rather than automatically integrating disparate workflow components.

K. Vampir

Vampir is a performance visualization tool designed for postmortem analysis of parallel program traces collected from large-scale HPC applications [49]. It provides an interactive GUI for exploring time-based behavior, communication, and synchronization events across thousands of processes or threads. Using traces generated by performance tools such as Score-P or TAU (through OTF/OTF2 formats), Vampir enables a detailed examination of computation–communication

overlap, load imbalance, and collective operation timing. Its timeline and statistical displays allow users to correlate events across multiple application components or workflow stages, giving a coarse workflow-level view when traces from distinct binaries are merged. However, Vampir does not natively represent workflow dependencies or DAG structures, and full cross-step workflow visualization requires external trace aggregation and annotation.

L. Jumpshot

Jumpshot is a postmortem performance visualization tool originally developed at Argonne National Laboratory for profiling parallel MPI and threaded applications [101]. It displays execution as timelines (Gantt charts) and histograms based on SLOG-2 or CLOG trace formats, enabling users to zoom into event sequences, detect anomalous durations, and view communication or computation phases over time. While Jumpshot excels at visualizing behavior within a single program's execution (especially communication and synchronization patterns), it is not inherently designed to aggregate or visualize performance data across multiple distinct workflow components or heterogeneous applications without manual integration.

M. Grafana

Grafana is an open-source visualization and analytics platform designed for real-time monitoring of metrics from diverse data sources, including Prometheus [102], InfluxDB, Elasticsearch, and proprietary telemetry systems [103]. It provides interactive dashboards, time series plots, and alerting capabilities that make it well suited for observing system performance and resource utilization. Grafana's plugin architecture enables integration with HPC monitoring stacks such as LDMS and Prometheus, allowing visualization across compute nodes, jobs, and applications. However, its focus is on metrics aggregation and system-level observability rather than program- or workflow-level performance analysis. While it can be configured to visualize workflow-related data through custom exporters or APIs, it lacks native support for linking fine-grained execution traces or profiling information across multiple workflow components.

N. Hatchet

Hatchet is a Python-based framework for analyzing hierarchical performance data collected from tools such as HPCToolkit, Caliper, or TAU [104], [105]. It provides a flexible data model for representing calling context trees (CCTs) or DAGs of performance metrics and supports powerful query, aggregation, and comparison operations through a Pandas-like API. Figure 12 shows the visualization supported by Hatchet. Hatchet's strength lies in post hoc analysis and data reduction across multiple runs or configurations, making it useful for comparative performance studies. However, it operates at the level of performance data extracted from individual applications rather than full workflow systems. Although it can

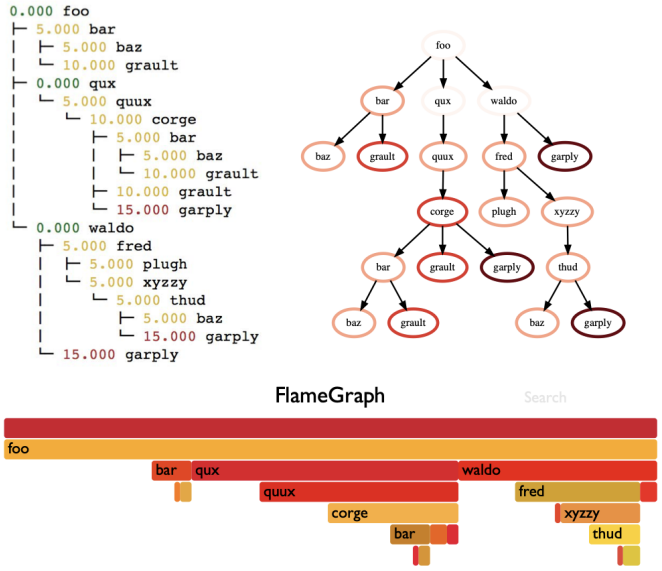


Fig. 12. Three different visualizations (the call graph, a tree-based call path, and a flame graph) generated by Hatchet for the same data [104]. These visualizations are well-suited for a single application or a single component of a workflow, as supported by Hatchet.

merge data from several program executions, it lacks inherent mechanisms to visualize or analyze performance across heterogeneous workflow components with distinct execution semantics, unless the data is manually integrated.

O. Thicket

Thicket is a Python-based framework built on top of Hatchet to enable scalable analysis of ensembles of performance profiles collected from multiple runs, configurations, or platforms [106]. It extends Hatchet's data model to represent collections of CCTs, supporting statistical analysis, correlation, and machine learning workflows on performance data. Thicket provides capabilities for grouping, filtering, and visualizing multi-run trends, making it particularly useful for comparative performance studies or identifying variability across large experiment sets. However, similar to Hatchet, Thicket operates at the application level and is not designed for workflow-level integration, where data from distinct, heterogeneous workflow components are automatically linked. Its visualization and analytics focus on ensembles of program executions rather than orchestrated workflow structures.

P. Drishti

Drishti [107]–[109] is an I/O performance visualization tool developed at Lawrence Berkeley National Laboratory to assist in the analysis of large-scale parallel I/O behavior. It provides a scalable and interactive environment for visualizing I/O traces collected from HPC applications, helping users identify access patterns, bottlenecks, and inefficiencies in parallel file system usage. Drishti focuses on presenting correlations between application-level events and I/O activity, making it easier to interpret complex performance data across

multiple layers of the I/O stack. Drishti provides fine-grained visualization of I/O traces from individual HPC applications, offering detailed intra-application insight into data movement and access patterns. While it can be used to compare I/O behavior across multiple runs, it is not inherently workflow-aware.

Key Takeaways

Traditional visualization tools are optimized for detailed, intra-application analysis of tightly coupled parallel programs, focusing on timelines, communication patterns, or per-rank metrics. However, complex scientific workflows introduce new challenges that extend visualization beyond individual applications. Workflow-level visualization must capture both intra-application behavior and inter-application relationships across heterogeneous components, execution environments, and time scales. It must integrate data from disparate sources, linking system-level and application-level metrics while preserving contextual information about data and control dependencies. Consequently, workflow-aware visualization tools prioritize scalability, data composability, interactivity, and cross-component correlation to provide holistic insight into end-to-end workflow behavior rather than just isolated application performance.

VI. Conclusion

Complex scientific workflows, integrated with artificial intelligence and machine learning, are becoming central to modern scientific research. Ensuring that these workflows utilize expensive HPC resources efficiently requires robust performance analysis and visualization. While numerous tools exist in the HPC ecosystem, most were originally designed for single, bulk-synchronous applications. Consequently, they fall short of addressing the performance and visualization needs of complex workflows that involve multiple interdependent parallel tasks and significant data movement. To support such workloads effectively, tools must extend their capabilities beyond per-task or per-application analysis to provide a holistic view of workflow execution and dependencies, enabling the identification of bottlenecks and critical paths.

This survey has reviewed state-of-the-art performance analysis and visualization tools, outlining their core characteristics and assessing their suitability for workflow-level performance evaluation. Our findings indicate that while these tools excel at fine-grained analysis, few support integrated coarse-grained visualization across entire workflows. This gap underscores the need for new tools, or extensions to existing ones, that explicitly target workflow-aware performance analysis.

Workflow Management Systems (WMS) play a complementary role by enabling workflow design, orchestration, and execution. However, workflow characteristics remain constrained by the underlying WMS. Some systems provide limited mechanisms for performance data collection, while others, such as Pegasus, have made progress toward holistic workflow visualization. These developments signal both a gap and an opportunity for deeper integration between WMS and performance analysis frameworks.

Looking forward, one promising direction is workflow performance prediction, particularly for sub-tasks that lie on the critical path. Given the high cost and setup time of scientific workflows, predicting the performance of workflow tasks with different configurations could significantly reduce experimentation overhead. Large Language Models (LLMs) offer a new avenue for such predictive modeling. Recent studies [110]–[113] have proposed and explored the extent to which LLMs can understand and reason about HPC performance behavior, reflecting a growing research interest in their potential for optimization and prediction. Incorporating these capabilities into workflow analysis pipelines could guide scientists toward optimal configurations before execution, ultimately accelerating discovery and improving resource efficiency.

References

[1] D. Qian, "High performance computing: a brief review and prospects," *National Science Review*, vol. 3, no. 1, p. 16—23, 2016.

[2] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17302510

[3] T. Ben-Nun, T. Gamblin, D. S. Hollman, H. Krishnan, and C. J. Newburn, "Workflows are the new applications: Challenges in performance, portability, and productivity," in *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2020, pp. 57–69.

[4] P. Krzyzanowski, "Bulk synchronous parallel and pregel," 2022, accessed: April 3, 2025. [Online]. Available: https://people.cs.rutgers.edu/~pxk/417/notes/pregel.html

[5] R. C. Calinescu, *The Bulk-Synchronous Parallel Model*. London: Springer London, 2000, pp. 5–12. [Online]. Available: https://doi.org/10.1007/978-1-4471-0763-7_2

[6] F. Ferrarotti, S. González, and K.-D. Schewe, "Bsp abstract state machines capture bulk synchronous parallel computations," *Sci. Comput. Program.*, vol. 184, no. C, Oct. 2019. [Online]. Available: https://doi.org/10.1016/j.scico.2019.102319

[7] C. Navarro, N. Hitschfeld, and L. Mateu, "A survey on parallel computing and its applications in data-parallel problems using gpu architectures," *Communications in Computational Physics*, vol. 15, pp. 285–329, 09 2013.

[8] I. Karlin, A. Bhatele, J. Keasler, B. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. Still, "Exploring traditional and emerging parallel programming models using a proxy application," 05 2013, pp. 919–932.

[9] T. Adefemi, "What every computer scientist needs to know about parallelization," 2025. [Online]. Available: https://arxiv.org/abs/2504.03647

[10] X. Zhao, M. Papagelis, A. An, B. X. Chen, J. Liu, and Y. Hu, "Elastic bulk synchronous parallel model for distributed deep learning," 2020. [Online]. Available: https://arxiv.org/abs/2001.01347

[11] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X08000861

[12] B. Ludäscher, S. Bowers, and T. McPhillips, *Scientific Workflows*. Boston, MA: Springer US, 2009, pp. 2507–2511. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1471

[13] S. B. Davidson and J. Freire, "Provenance and scientific workflows: challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: Association for Computing

Machinery, 2008, p. 1345–1350. [Online]. Available: https://doi.org/10.1145/1376616.1376772

[14] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018. [Online]. Available: https://doi.org/10.1177/1094342017704893

[15] 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

[16] A. Al-Saadi, D. H. Ahn, Y. Babuji, K. Chard, J. Corbett, M. Hategan, S. Herbein, S. Jha, D. Laney, A. Merzky, T. Munson, M. Salim, M. Titov, M. Turilli, T. D. Uram, and J. M. Wozniak, "Exaworks: Workflows for exascale," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021, pp. 50–57.

[17] V. Cuevas-Vicenttín, S. Dey, S. Köhler, S. Riddle, and B. Ludäscher, "Scientific workflows and provenance: Introduction and research opportunities," *Datenbank Spektrum*, vol. 12, no. 4, pp. 193–203, 2012. [Online]. Available: https://link.springer.com/article/10.1007/s13222-012-0100-z

[18] F. Di Natale, H. Bhatia, T. S. Carpenter, C. Neale, S. Kokkila-Schumacher, T. Oppelstrup, L. Stanton, X. Zhang, S. Sundram, T. R. W. Scogland, G. Dharuman, M. P. Surh, Y. Yang, C. Misale, L. Schneidenbach, C. Costa, C. Kim, B. D'Amora, S. Gnanakaran, D. V. Nissley, F. Streitz, F. C. Lightstone, P.-T. Bremer, J. N. Glosli, and H. I. Ingólfsson, "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*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356197

[19] D. H. Ahn, X. Zhang, J. Mast, S. Herbein, F. Di Natale, D. Kirshner, S. A. Jacobs, I. Karlin, D. J. Milroy, B. De Supinski, B. Van Essen, J. Allen, and F. C. Lightstone, "Scalable composition and analysis techniques for massive scientific workflows," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 32–43.

[20] H. Bhatia, T. A. Patki, S. Brink, L. Pottier, T. M. Stitt, K. Parasyris, D. J. Milroy, D. E. Laney, R. C. Blake, J.-S. Yeom, P.-T. Bremer, and C. Doutriaux, "Autonomous multiscale library," Accessed: 2023-10-15, 2023, version 1.0.0. [Online]. Available: https://github.com/LLNL/AMS

[21] J. L. Peterson, T. Bender, R. Blake, N.-Y. Chiang, M. G. Fernández-Godino, B. Garcia, A. Gillette, B. Gunnarson, C. Hansen, J. Hill *et al.*, "Toward digital design at the exascale: An overview of project icecap," *Physics of Plasmas*, vol. 31, no. 6, 06 2024. [Online]. Available: https://www.osti.gov/biblio/2407209

[22] R. S. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi, "Cybershake: A physics-based seismic hazard model for southern california," *Pure and Applied Geophysics*, vol. 168, no. 3, pp. 367–381, 2010. [Online]. Available: https://doi.org/10.1007/s00024-010-0161-6

[23] M. Rynge, G. Juve, J. Kinney, J. Good, B. Berriman, A. Merrihew, and E. Deelman, "Producing an infrared multiwavelength galactic plane atlas using montage, pegasus, and amazon web services," in *Astronomical Data Analysis Software and Systems XXII*, M. W. Wertheimer and D. A. Bohlender, Eds., vol. 475. Astronomical Society of the Pacific, 2013, pp. 345–348. [Online]. Available: https://pegasus.isi.edu/wordpress/wp-content/papercite-data/pdf/rynge2013adass.pdf

[24] T. A. Desautels, K. T. Arrildt, A. T. Zemla, E. Y. Lau, F. Zhu, D. Ricci, S. Cronin, S. J. Zost, E. Binshtein, S. M. Scheaffer, B. Dadonaite, B. K. Petersen, T. B. Engdahl, E. Chen, L. S. Handal, L. Hall, J. W. Goforth, D. Vashchenko, S. Nguyen, D. R. Weilhammer, J. K.-Y. Lo, B. Rubinfeld, E. A. Saada, T. Weisenberger, T.-H. Lee, B. Whitener, J. B. Case, A. Ladd, M. S. Silva, R. M. Haluska, E. A. Grzesiak, C. G. Earnhart, S. Hopkins, T. W. Bates, L. B. Thackray, B. W. Segelke, A. M. Lillo, S. Sundaram, J. Bloom, M. S. Diamond, J. E. Crowe, R. H. Carnahan, and D. M. Faissol, "Computationally restoring the potency of a clinical antibody against sars-cov-2 omicron subvariants," *bioRxiv*, 2023. [Online]. Available: https://www.biorxiv.org/content/early/2023/04/24/2022.10.21.513237

[25] T. Coleman, H. Casanova, L. Pottier, M. Kaushik, E. Deelman, and R. F. da Silva, "Wfcommons: A framework for enabling scientific workflow research and development," 2021. [Online]. Available: https://arxiv.org/abs/2105.14352

[26] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: Moving across paradigms," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016. [Online]. Available: https://doi.org/10.1145/3012429

[27] M. Atkinson, S. Gesing, J. Montagnat, and I. Taylor, "Scientific workflows: Past, present and future," *Future Generation Computer Systems*, vol. 75, pp. 216–227, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17311202

[28] Unknown, "Workflow Patterns," http://www.workflowpatterns.com, 2010-2023.

[29] 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.

[30] P. Krawczuk, G. Papadimitriou, R. Tanaka, T. M. Anh Do, S. Subramanya, S. Nagarkar, A. Jain, K. Lam, A. Mandal, L. Pottier, and E. Deelman, "A performance characterization of scientific machine learning workflows," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021, pp. 58–65.

[31] R. Ferreira da Silva, H. Casanova, K. Chard, I. Altintas, B. Balis, T. Coleman, F. Coppens, F. Natale, B. Enders, T. Fahringer, R. Filgueira, G. Fursin, D. Garijo, C. Goble, D. Howell, S. Jha, D. Katz, D. Laney, and M. Wolf, "A community roadmap for scientific workflows research and development," 11 2021, pp. 81–90.

[32] R. F. da Silva, "MAGIC: Machine Learning Guided Scientific Discovery," 2023, presentation at NITRD MAGIC, May 3, 2023. [Online]. Available: https://www.nitrd.gov/nitrdgroups/images/2023/MAGIC-Rafael-Ferreira-da-Silva-05032023.pdf

[33] R. M. Badia, L. Berti-Equille, R. F. da Silva, and U. Leser, "Integrating HPC, AI, and Workflows for Scientific Data Analysis (Dagstuhl Seminar 23352)," *Dagstuhl Reports*, vol. 13, no. 8, pp. 129–164, 2024. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/DagRep.13.8.129

[34] S. Jha, V. R. Pascuzzi, and M. Turilli, "Ai-coupled hpc workflows," 2022. [Online]. Available: https://arxiv.org/abs/2208.11745

[35] F. Suter, T. Coleman, İlkay Altintaş, R. M. Badia, B. Balis, K. Chard, I. Colonnelli, E. Deelman, P. Di Tommaso, T. Fahringer, C. Goble, S. Jha, D. S. Katz, J. Köster, U. Leser, K. Mehta, H. Oliver, J.-L. Peterson, G. Pizzi, L. Pottier, R. Sirvent, E. Suchyta, D. Thain, S. R. Wilkinson, J. M. Wozniak, and R. Ferreira da Silva, "A terminology for scientific workflow systems," *Future Generation Computer Systems*, vol. 174, p. 107974, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X25002699

[36] N. Dube, D. Roweth, P. Faraboschi, and D. Milojicic, "Future of hpc: The internet of workflows," *IEEE Internet Computing*, vol. 25, no. 5, pp. 26–34, 2021.

[37] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[38] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, 03 2015.

[39] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[40] D. Gunter, B. Tierney, and et al., "Online workflow management and performance analysis with the pegasus workflow management system," in *Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM'11)*, 2011, pp. 988–995.

[41] G. Papadimitriou, C. Wang, K. Vahi, R. Ferreira da Silva, A. Mandal, Z. Liu, R. Mayani, M. Rynge, M. Kiran, V. Lynch, R. Kettimuthu, E. Deelman, J. Vetter, and I. Foster, "End-to-end online performance data capture and analysis for scientific workflows," *Future Generation Computer Systems*, vol. 117, pp. 170–185, 2021.

[42] Pegasus Team, "Pegasus workflow management system documentation," https://pegasus.isi.edu/documentation/, 2025, accessed: 2025-01-22.

[43] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[44] Nextflow. Reports — nextflow documentation. Accessed: 2025-10-22. [Online]. Available: https://www.nextflow.io/docs/latest/reports.html

[45] F. Mölder, K. P. Jablonski, B. Letcher, M. B. Hall, C. H. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. O. Twardziok, A. Kanitz, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen, and J. Köster, "Sustainable data analysis with snakemake," *F1000Research*, vol. 10, p. 33, 2021.

[46] Snakemake Development Team, "Snakemake documentation," https://snakemake.readthedocs.io/, accessed: 2025-10-22.

[47] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[48] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the kepler scientific workflow system," in *Provenance and Annotation of Data*, ser. IPAW 2006, Lecture Notes in Computer Science, vol. 4145. Springer, 2006, pp. 118–132.

[49] W. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, "Vampir: Visualization and analysis of mpi resources," *Supercomputer*, vol. 12, 05 1996.

[50] D. Computadors, V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," *WoTUG-18*, vol. 44, 03 1995.

[51] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702–719, Apr. 2010.

[52] P. Mucci, S. Moore, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," 01 1999.

[53] D. Terpstra, H. Jagode, H. You, and J. Dongarra, *Collecting performance data with PAPI-C*, 05 2010, pp. 157–173.

[54] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 154–165.

[55] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[56] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya, and S. Cranford, "Open—speedshop: An open source infrastructure for parallel performance analysis," *Scientific Programming*, vol. 16, no. 2-3, 12 2007. [Online]. Available: https://www.osti.gov/biblio/1198001

[57] S. S. Shende and A. D. Malony, "The tau parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006. [Online]. Available: https://doi.org/10.1177/1094342006064482

[58] A. Knüpfer, C. Rössel, D. a. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir," in *Tools for High Performance Computing 2011*, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 79–91.

[59] J. R. Madsen, M. G. Awan, H. Brunie, J. Deslippe, R. Gayatri, L. Oliker, Y. Wang, C. Yang, and S. Williams, "Timemory: Modular performance analysis for hpc," in *High Performance Computing*, P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, Eds. Cham: Springer International Publishing, 2020, pp. 434–452.

[60] D. Boehme, P. Aschwanden, O. Pearce, K. Weiss, and M. LeGendre, "Ubiquitous Performance Analysis," in *High Performance Computing*, B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, Eds. Cham: Springer International Publishing, 2021, pp. 431–449.

[61] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz, "Caliper: Performance Introspection for HPC Software Stacks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. IEEE Press, 2016.

[62] D. Yokelson, M. Titov, S. Ramesh, O. Kilic, M. Turilli, S. Jha, and A. Malony, "Enabling performance observability for heterogeneous hpc workflows with soma," in *Proceedings of the 53rd International Conference on Parallel Processing*, ser. ICPP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 220–230. [Online]. Available: https://doi.org/10.1145/3673038.3673100

[63] A. Malony and S. Shende, "Performance technology for complex parallel and distributed systems," *Scalable Computing: Practice and Experience*, vol. 4, 01 2001.

[64] K. A. Huck, S. Shende, A. D. Malony, C. Coti, W. Spear, J. Alcaraz, D. Yokelson, S. Ramesh, M. M. A. Haque, C. Wood, N. Chaimov, C. Durbin, A. Johnson, J. Lambert, and I. Beekman, "Preparing the tau performance system for exascale and beyond," *The International Journal of High Performance Computing Applications*, vol. 0, no. 0, p. 10943420251334456, 0. [Online]. Available: https://doi.org/10.1177/10943420251334456

[65] C. Xie and W. Xu, "Performance visualization for tau instrumented scientific workflows." Brookhaven National Lab. (BNL), Upton, NY (United States), 01 2018. [Online]. Available: https://www.osti.gov/biblio/1560001

[66] N. R. Tallent, J. Mellor-Crummey, M. Franco, R. Landrum, and L. Adhianto, "Scalable fine-grained call path tracing," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 63–74. [Online]. Available: https://doi.org/10.1145/1995896.1995908

[67] L. Adhianto, J. Anderson, R. M. Barnett, D. Grbic, V. Indic, M. Krentel, Y. Liu, S. Milaković, W. Phan, and J. Mellor-Crummey, "Refining hpctoolkit for application performance analysis at exascale," *The International Journal of High Performance Computing Applications*, vol. 38, no. 6, pp. 612–632, 2024. [Online]. Available: https://doi.org/10.1177/10943420241277839

[68] A. Gocht, R. Schöne, and J. Frenzel, "Advanced python performance monitoring with score-p," *CoRR*, vol. abs/2010.15444, 2020. [Online]. Available: https://arxiv.org/abs/2010.15444

[69] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. Nagel, and F. Wolf, "Open trace format 2: The next generation of scalable trace formats and support libraries," vol. 22, 01 2012, pp. 481 – 490.

[70] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya, and S. Cranford, "Analyzing the performance of scientific applications with openspeedshop." Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 10 2009. [Online]. Available: https://www.osti.gov/biblio/967760

[71] NASA, "Using MPIProf for Performance Analysis," https://www.nas.nasa.gov/hecc/support/kb/using-mpiprof-for-performance-analysis_525.html?utm_source=chatgpt.com.

[72] D. Marques, H. Duarte, A. Ilic, L. Sousa, R. Belenov, P. Thierry, and Z. A. Matveev, "Performance analysis with cache-aware roofline model in intel advisor," in *2017 International Conference on High Performance Computing and Simulation (HPCS)*, 2017, pp. 898–907.

[73] NVIDIA Corporation, "Nsight systems user guide," https://docs.nvidia.com/nsight-systems/, accessed: 2025-09-28.

[74] N. Ding, B. Austin, Y. Liu, N. Mehta, S. Farrell, J. P. Blaschke, L. Oliker, H. A. Nam, N. J. Wright, and S. Williams, "A workflow roofline model for end-to-end workflow performance analysis," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '24. IEEE Press, 2024. [Online]. Available: https://doi.org/10.1109/SC41406.2024.00071

[75] A. Lisan, T. Patki, S. Brink, K. Parasyris, B. Gunnarson, G. Georgakoudis, and H. Childs, "Enabling lightweight performance analysis of complex scientific workflows with perfflowaspect," in *Proceedings of the 37th International Conference on Scalable Scientific Data Management*, ser. SSDBM '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3733723.3733734

[76] H. Ather, J. L. Bez, C. Wang, H. Childs, A. D. Malony, and S. Byna, "Parallel i/o characterization and optimization on large-scale hpc systems: A 360-degree survey," 2024. [Online]. Available: https://arxiv.org/abs/2501.00203

[77] D. Król, R. F. da Silva, E. Deelman, and V. E. Lynch, "Workflow performance profiles: Development and analysis," in *Euro-Par 2016: Parallel Processing Workshops*, F. Desprez, P.-F. Dutot, C. Kaklamanis,

L. Marchal, K. Molitorisz, L. Ricci, V. Scarano, M. A. Vega-Rodríguez, A. L. Varbanescu, S. Hunold, S. L. Scott, S. Lankes, and J. Weidendorfer, Eds. Cham: Springer International Publishing, 2017, pp. 108–120.

[78] R. Tschueter, C. Herold, W. Williams, M. Knespel, and M. Weber, "A top-down performance analysis methodology for workflows: Tracking performance issues from overview to individual operations," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, 2019, pp. 21–30.

[79] D. Gunter, E. Deelman, T. Samak, C. H. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, and K. Vahi, "Online workflow management and performance analysis with stampede," in *2011 7th International Conference on Network and Service Management*, 2011, pp. 1–10.

[80] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee, "Netlogger: a toolkit for distributed system performance analysis," in *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728)*, 2000, pp. 267–273.

[81] A. Lisan, T. Patki, S. Brink, Z. Yang, S. Greene, K. Parasyris, S. Taneja, and H. Childs, "Perfflowaspect: A user-friendly performance tool for scientific workflows," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC24)*, 2024, poster. [Online]. Available: https://sc24.supercomputing.org/proceedings/poster/poster_files/post223s2-file3.pdf

[82] LLNL, "PerfFlowAspect," https://github.com/flux-framework/PerfFlowAspect, May 2023.

[83] C. Lattner and V. Adve, "Llvm: a compilation framework for lifelong program analysis & transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75–86.

[84] D. Yokelson, "In situ visualization of performance data for high-performance computing applications," Department of Computer Science, University of Oregon, Technical Report AREA-202206, 2022. [Online]. Available: https://www.cs.uoregon.edu/Reports/AREA-202206-Yokelson.pdf

[85] C. Kelly, S. Ha, K. Huck, H. Van Dam, L. Pouchard, G. Matyasfalvi, L. Tang, N. D'Imperio, W. Xu, S. Yoo, and K. K. Van Dam, "Chimbuko: A workflow-level scalable performance trace analysis tool," in *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ser. ISAV'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 15–19. [Online]. Available: https://doi.org/10.1145/3426462.3426465

[86] H. Devarajan, L. Pottier, K. Velusamy, H. Zheng, I. Yildirim, O. Kogiou, W. Yu, A. Kougkas, X.-H. Sun, J. S. Yeom, and K. Mohror, "Dftracer: An analysis-friendly data flow tracer for ai-driven workflows," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '24. IEEE Press, 2024. [Online]. Available: https://doi.org/10.1109/SC41406.2024.00023

[87] C. Wang, I. Yildirim, H. Devarajan, K. Mohror, and M. Snir, "Recorder: Comprehensive parallel i/o tracing and analysis," *arXiv preprint arXiv:2501.04654*, 2025. [Online]. Available: https://arxiv.org/abs/2501.04654

[88] R. Wang, S. Snyder, D. Benjamin, Z. Dong, P. Gartung, and K. Herner, "Darshan for hep applications," vol. 295. Brookhaven National Laboratory (BNL), Upton, NY (United States); Fermi National Accelerator Laboratory (FNAL), Batavia, IL (United States); Argonne National Laboratory (ANL), Argonne, IL (United States), 12 2023. [Online]. Available: https://www.osti.gov/biblio/2468766

[89] 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.

[90] K. Isaacs, A. Gimenez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization," 06 2014.

[91] E. Deelman, K. Vahi, M. Rynge, G. Juve, R. Mayani, and R. F. da Silva, "Pegasus in the cloud: Science automation through workflow technologies," *IEEE Internet Computing*, vol. 20, no. 1, pp. 70–76, 2016.

[92] D. O'Leary, T. Mattson, H. Cui, K. Raman, J. Rego, and A. Rodriguez, "Intel advisor roofline analysis: A new way to visualize performance on intel cpus and gpus," in *Proceedings of the IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2018. [Online]. Available: https://cdrdv2-public.intel.com/671165/roofline-analysis-with-intel-advisor.pdf

[93] Intel Corporation, "Intel vtune profiler performance analysis user guide," https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide/, 2024, accessed: 2025-10-26.

[94] NVIDIA Corporation, "Nvidia nsight compute user guide," 2024, version 2024.2. [Online]. Available: https://docs.nvidia.com/nsight-compute/

[95] S. Jones, J. Larkin, and N. Wilt, "Nvidia nsight compute: An interactive kernel profiler for cuda applications," NVIDIA GPU Technology Conference (GTC) Presentation, NVIDIA Corporation, 2019. [Online]. Available: https://developer.nvidia.com/nsight-compute

[96] N. Jain, J. Larkin, and S. Jones, "Nvidia nsight systems: A system-wide performance analysis tool for heterogeneous computing," NVIDIA GPU Technology Conference (GTC) Presentation, NVIDIA Corporation, 2020. [Online]. Available: https://developer.nvidia.com/nsight-systems

[97] R. Bell, A. Malony, and S. Shende, "Paraprof: A portable, extensible, and scalable tool for parallel performance profile analysis," vol. 2790, 08 2003, pp. 17–26.

[98] K. Huck and A. Malony, "Perfexplorer: A performance data mining framework for large-scale parallel computing," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005, pp. 41–41.

[99] Google LLC, "Perfetto documentation," https://perfetto.dev/docs, 2024, accessed: October 2025.

[100] K. E. Isaacs, A. G. Landge, T. Gamblin, P.-T. Bremer, V. Pascucci, and B. Hamann, "Abstract: Exploring performance data with boxfish," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 1380–1381.

[101] O. Zaki, E. Lusk, W. Gropp, and D. Swider, "Toward scalable performance visualization with jumpshot," *Int. J. High Perform. Comput. Appl.*, vol. 13, no. 3, p. 277–288, Aug. 1999. [Online]. Available: https://doi.org/10.1177/109434209901300310

[102] Grafana Labs, "What is prometheus? — grafana documentation," https://grafana.com/docs/grafana/latest/fundamentals/intro-to-prometheus/, 2025, accessed: November 3, 2025.

[103] ——, "Grafana documentation," https://grafana.com/docs/grafana/latest/, 2025, accessed: November 2, 2025.

[104] A. Bhatele, S. Brink, and T. Gamblin, "Hatchet: pruning the overgrowth in parallel profiles," 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/10.1145/3295500.3356219

[105] I. Lumsden, J. Luettgau, V. Lama, C. Scully-Allison, S. Brink, K. E. Isaacs, O. Pearce, and M. Taufer, "Enabling call path querying in hatchet to identify performance bottlenecks in scientific applications," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 256–266.

[106] O. Pearce, J. Burmark, R. Hornung, B. Bogale, I. Lumsden, M. McKinsey, D. Yokelson, D. Boehme, S. Brink, M. Taufer, and T. Scogland, "Raja performance suite: Performance portability analysis with caliper and thicket," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1206–1218.

[107] J. L. Bez, H. Ather, and S. Byna, "Drishti: Guiding end-users in the i/o optimization journey," in *2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW)*, 2022, pp. 1–6.

[108] H. Ather, J. L. Bez, Y. Xia, and S. Byna, "Drilling down i/o bottlenecks with cross-layer i/o profile exploration," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 532–543.

[109] H. Ather, J. L. Bez, B. Norris, and S. Byna, "Illuminating the i/o optimization path of scientific applications," in *High Performance Computing*, A. Bhatele, J. Hammond, M. Baboulin, and C. Kruse, Eds. Cham: Springer Nature Switzerland, 2023, pp. 22–41.

[110] B. Cui, T. Ramesh, O. Hernandez, and K. Zhou, "Do large language models understand performance optimization?" 2025. [Online]. Available: https://arxiv.org/abs/2503.13772

[111] G. Bolet, G. Georgakoudis, H. Menon, K. Parasyris, N. Hasabnis, H. Estes, K. Cameron, and G. Oren, "Can large language models predict parallel code performance?" in *Proceedings of the*

34th International Symposium on High-Performance Parallel and Distributed Computing, ser. HPDC '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3731545.3743645

[112] D. Nichols, A. Marathe, H. Menon, T. Gamblin, and A. Bhatele, "Hpc-coder: Modeling parallel programs using large language models," in *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, 2024, pp. 1–12.

[113] H. B. Ather, C. Wang, H. Devarajan, H. Childs, and K. Mohror, "Smartio: A lightweight end-to-end workflow for runtime i/o optimization of hpc systems," in *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC Workshops '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1396–1405. [Online]. Available: https://doi.org/10.1145/3731599.3767509