2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)

Toward Understanding I/O Behavior in HPC Workflows

Jakob Lüttgau[†*], Shane Snyder[*], Philip Carns[*], Justin M. Wozniak[*], Julian Kunkel[‡], Thomas Ludwig[†]
[*]Argonne National Laboratory (ANL), [†]German Climate Computing Center (DKRZ), [‡]University of Reading
luettgau@dkrz.de

***Abstract*—Scientific discovery increasingly depends on complex workflows consisting of multiple phases and sometimes millions of parallelizable tasks or pipelines. These workflows access storage resources for a variety of purposes, including preprocessing, simulation output, and postprocessing steps. Unfortunately, most workflow models focus on the scheduling and allocation of computational resources for tasks while the impact on storage systems remains a secondary objective and an open research question. I/O performance is not usually accounted for in workflow telemetry reported to users.**

In this paper, we present an approach to augment the I/O efficiency of the individual tasks of workflows by combining workflow description frameworks with system I/O telemetry data. A conceptual architecture and a prototype implementation for HPC data center deployments are introduced. We also identify and discuss challenges that will need to be addressed by workflow management and monitoring systems for HPC in the future. We demonstrate how real-world applications and workflows could benefit from the approach, and we show how the approach helps communicate performance-tuning guidance to users.

I. Introduction

Supercomputing resources enable scientific discovery at an unprecedented pace. Scientists across domains employ complex data-processing pipelines and workflows to automate the evaluation of experiments and simulation results. Along with the execution of these workflows, huge amounts of data need to be handled by HPC I/O subsystems. Unfortunately, compute and storage performance capabilities are on divergent trajectories. While compute capabilities scale relatively well because of massive use of distributed computing and specialized compute hardware [1], storage systems trying to match required performance characteristics are usually constrained by a lack of suitable and affordable technologies, as well as limited energy envelopes. In addition, storage systems tend to be shared by multiple users, and they employ complex memory hierarchies to balance cost and performance. The resulting software to drive storage and memory stacks evolves slowly, and new approaches require time to prove that they are stable enough to be used in production.

As exascale systems are about to become a reality, I/O bottlenecks are a growing concern that needs to be addressed. Overcoming the challenges raised requires changes throughout the storage stacks, and hence many research efforts are integrated into larger codesign initiatives [2] [3] [4]. At the same time, design decisions for large-scale storage systems are influenced by market forces, with future system architectures [5] trying to make use of commoditization and employing deep memory hierarchies to conserve cost.

The confluence of system complexity, application optimizations, and market forces lead to emergent I/O subsystem behavior that is often not well understood. Systems remain below theoretic peak performance since different usage patterns can have unforeseen side effects on the system. In order to better understand this behavior, monitoring tools are being developed and deployed to allow systemwide monitoring in modern HPC systems. Recent work has shown that the value of I/O telemetry is greatly increased if it is analyzed in context with broader contextual information [6], though this type of analysis is generally performed only after the fact.

An untapped potential for increasing our contextual understanding of I/O behavior exists by exploiting knowledge about the structure of HPC workflows. Since what is meant by HPC workflows is not always obvious, a definition useful from a storage perspective is offered in Section II-A. That workflow knowledge is not routinely considered by decision components across the storage stack can be attributed to a number of factors. On the one hand, monitoring and telemetry information are not always collected. On the other hand, application developers lack incentive to switch to workflow tools, especially when workflow models might impose constraints on or add additional effort to a project.

In a mid- to long-term perspective, however, users specifying their workflows help enable a number of on-going research efforts. With a workflow description one can more easily associate monitoring and telemetry data with the workflow. More distant is the integration of this information into resource allocation and scheduling systems. Data about how workflows utilize the storage systems will eventually help develop and train adaptive and more intelligent systems.

In this paper we present an approach to attribute I/O activity to task and data objects in HPC workflows. Our contributions are as follows:

1) An architecture proposal for a holistic association of I/O activity with task and data of scientific workflows
2) A prototype implementation to demonstrate the potential of an independent set of tools to work with different workflow management systems
3) Tools for analysis and visualization of workflows for I/O researchers as well as for communication with users of supercomputing resources.

978-1-7281-0192-7/18/$31.00 ©2018 IEEE
DOI 10.1109/PDSW-DISCS.2018.00010

The remainder of the paper is structured as follows. Section II introduces related work and background on workflows, monitoring, and I/O. Standing challenges and an architecture necessary to address them are proposed in Section III. In Section IV an overview of the proof of concept implementation is presented. Section V discusses two use cases to utilize insight on workflow I/O in adaptive systems. Section VI concludes the paper with a summary and provides an outlook on future work.

II. Background & Related Work

This section introduces background information on state-of-the-art solutions to I/O activity capture and monitoring (Section II-D). The section includes a workflow definition useful for HPC workflows that need to make use of large-scale storage systems and parallel file systems in particular (Section II-A).

A. I/O Perspective on Scientific Workflows

Scientists commonly perform experiments on supercomputers using some notion of a workflow. Often, these workflows may not be explicitly specified in a form easily accessible to machines but instead are presented as a high-level description in a project proposal or as sets of scripts used by researchers. For complex workflows, especially when repeated many times, researchers are more likely to opt for workflow management systems (WMS) or workflow engines. A workflow management system provides and implements a task or data model requiring users to merely define the relationships, while the workflow engine executes the workflow with potential to transparently optimize resource utilization. Besides convenience for users, workflows offer opportunities to better anticipate future activity, including activity that will affect storage systems.

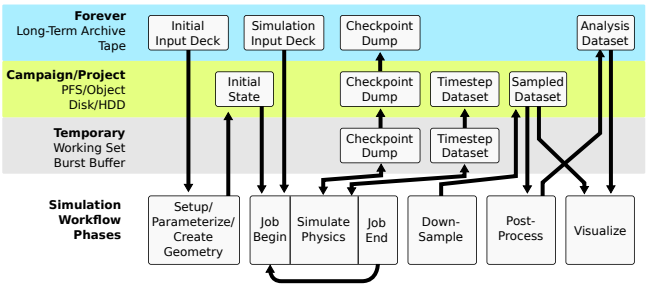
As no consistent universal definition for workflows holds over time and across different fields, this section briefly offers a definition that aims to be useful for workflows in HPC environments with storage systems in mind. To an extent, the definition leans on typical HPC workflows identified and published in an APEX whitepaper on workflows [7]. The perspective on how scientific workflows are expected to evolve for next-generation workflows and systems is supported by [8].

Most commonly, a *workflow* describes a number of *tasks* that should be performed to achieve a higher-level goal. Tasks usually will consume or yield a piece of *data*, which in many cases leads to an order in which tasks need to executed. Also worth considering are *pipelines*, since one commonly sees the same series of steps being performed on different input data, which in many cases give rise to parallel execution.

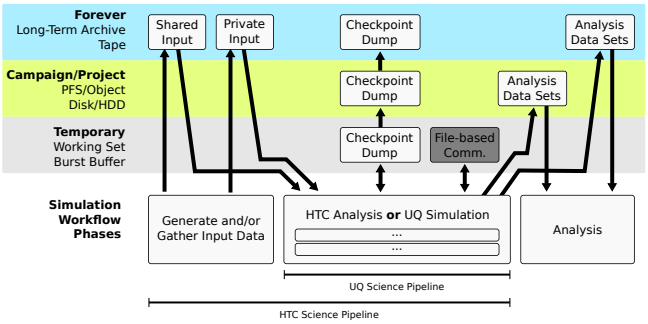
a) Task: A task is a logical entity or a program that can consume and/or produce data. Granularity may vary depending on the abstraction used by different workflow engines, with a single task potentially mapping to to a job, a process, a thread, or even a function call.

b) Data: Data comprises information that is retained or passed between tasks. Depending on the lifetime, data can be messages, database entries, or files and objects on a storage system. The granularity of data objects varies considerably between different workflow engines. Many workflow engines use notions of data sets in a hierarchical namespace, while others directly assume the presence of file systems. In some extreme cases, workflow engines might consider individual variables as small as a single byte or integer as data objects.

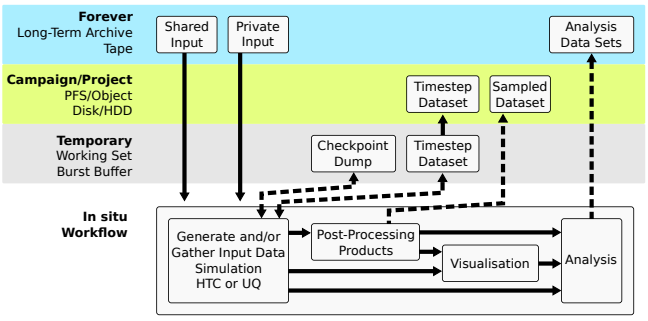
c) Workflow: A workflow governs the dependencies of tasks and data and is often represented by using a graph, with nodes being used to model tasks while edges represent dependencies. In many cases this representation will result in a directed acyclic graph (DAG), although complex workflows may contain cycles. Also, considering I/O activity in such a graph can lead to situations where data is being read and



(a) Stereotypical simulation workflow (see APEX Workflows [7])



(b) Stereotypical HQ/HTC workflow (see APEX Workflows [7])



(c) Approximation of an in situ workflow

Fig. 1: Abstractions of predominant existing and envisioned workflows in HPC environments according to [7] and [8]. The abstracted simulation, UQ, and HTC workflows are derived from [7]. In situ workflows are anticipated but not widely spread at this time as frameworks are in their early stages and adapting existing applications is not always straightforward.

written in ways that turn an acyclic task dependency graph into a cyclic task/data dependency graph. Implementation logic of tasks is often strictly separated (e.g., a binary) from the declaration of relationships, while fine-grained workflow engines seem to favor workflow definitions that allow mixing the two.

d) Pipeline: Some workflows feature a repeated chain of steps (which might include branching). If a workflow starts this same chain of steps, we refer to it as a pipeline. For example, depending on an input set for the workflow, every element in an array might require processing as an instance of a pipeline.

B. Common HPC Workflows

In order to address challenges of current and future workflows, this section introduces workflows commonly observed [7] in HPC environments. An outlook on anticipated workflows and recognized challenges that WMS will need to address before they are suitable for widespread adoption in HPC environments is provided by [8]. To a large extent, workflows depend on the facilities, instruments, hardware, and services scientists find in their respective institutions. Workflows can span different timespans from minutes up to years depending on their alignment to a given project. Nonetheless, a number of common patterns for workflows on HPC have emerged.

a) Simulation: Simulation workflows tend to consist of 3–4 major phases. An illustration of a stereotypical simulation workflow is given in Figure 1a. In the preprocessing phase, input data needs to be transformed into an initial state for the simulation. In the simulation/data generation phase, applications write snapshots and timestep data for fault tolerance, postprocessing applications, and potential archival. The postprocessing and visualization phases are deriving data products that in many cases may be published and preserved. A simulation workflow as outlined here might be part of the pipeline of other more data-intensive workflows.

b) Uncertainty Quantification: To deal with uncertainty when working with nonlinear or chaotic systems (as in many physical simulations), researchers commonly design workflows for uncertainty quantification (UQ). An illustration of a stereotypical UQ workflow is given in Figure 1b. Such workflows often consist of a large number of independent pipelines (typically with simulations running for multiple hours) that can be executed in parallel. Results from these parallel executions (often called ensembles) are then combined into analysis data sets.

c) High-Throughput Computing: Similar to UQ workflows are high-throughput computing (HTC) workflows, again illustrated in Figure 1b. In HTC a large number of pipelines are executed, but typically featuring tasks only of limited runtime/data volume. Often the objective is to explore a parameter space, for example, to find hyper-parameters in machine learning applications. From a storage system perspective, high-throughput computing is often associated with many relatively small files.

d) In Situ / Integrated Approaches: In situ workflows integrate their numerous distinct phases into a single application. Traditionally, these types of workflows have been implemented in the context of MPI. An illustration of a possible in situ workflow is given in Figure 1c. Such workflows allow for conservation of resources by exploiting data locality and avoiding overheads from resource allocation, context switching, and application startup/shutdown. As far as storage systems are concerned, the requirement to occasionally consume large amounts of snapshot data remains; but overall in situ workloads promise to reduce the load across network and storage systems.

e) Hybrid Approaches: Workflows will continue to evolve, and in many cases scientists will mix workflow models that are not easily integrated or solved by a single solution. Since collaborations in projects often span multiple locations and organizations, one must assume that different workflow tools being used even within a single project. Consequently, tools to analyze workflows should address this heterogeneity, for example by offering abstractions and modularity to integrate other tools and WMS with reasonable effort.

C. Workflow Engines for Automation in HPC Environments

TABLE I: Workflow Management Systems for HPC

Workflow Engine	Modes	Data Models	Graph	Telemetry	I/O
Swift/T [9], [10]	Job/Runtime	DSL	(✓)	×	×
Cylc [11]	(Dist.) Jobs	(✓)	✓	✓	×
Apache Spark [12]	Runtime	RDD	✓	✓	(✓)
Fireworks [13]	Distributed	-	✓	✓	(×)
Pegasus [14]	(Dist.) Jobs	-	✓	✓	×
TaskFarmer [15]	Commands	Files/Shards	×	×	×
Tigres [16]	Runtime	"Inputs"	-	✓	×
Ophidea [17]	Runtime	Datasets	✓	✓	×
Kepler [18]	Runtime	(✓)	✓	-	×

(×)/(✓): Restrictions Apply, -: No documentation found
DSL: Domain Specific Language, RDD: Resilient Distributed Datasets

As scientists continue to add complexity to their workflows and strive for reproducible scientific results, they demand more elaborate tools and frameworks that can automate the execution of workflows. Workflow management systems have gained considerable popularity with an increase in big data applications and have also become increasingly popular with users in the HPC community. Since some of the tools popular in a big data context are challenging to deploy in HPC systems, a number of WMS designed specifically for use on HPC systems are available and under active development. To an extent, this fragmentation can be attributed to the slightly different priorities from one scientific domain to another. Table I provides an overview of HPC WMS as well as a comparison of features relevant to an I/O perspective. In particular, since this work introduces tools to visualize I/O activity of workflows, we indicate whether WMS provide graphical representations to inspect or even edit workflows. Tools for visualization are provided by Cylc [11], Fireworks [13], Ophidea [17], Kepler [18], and Pegasus [14]. Kepler [18] is notable because an interactive graphical user interface is provided that also allows direct inspection of results. Cylc [11] uses the dot format to visualize workflows, which allows for easy customization.

For the prototype implementation of our workflow analysis tools, we started with support for two workflow engines that reside at two ends of the spectrum for workflow granularity. Swift/T implements a fine-grained and integrated approach to workflows that is anticipated to be increasingly relevant as applications strive to exploit exascale systems. Cylc follows a more traditional and distributed workflow approach and assumes the submission of jobs to batch scheduling systems. As explained in Section II-B, we anticipate that both models will remain relevant, and we therefore pursue an abstraction that accommodates both. The following two paragraphs provide a more in-depth description of Swift and Cylc.

a) Swift/T: Swift provides a domain-specific language with many features of a generic programming language to specify workflows. Swift has two different runtimes to interpret and execute a workflow description: Swift/K [19] and Swift/T [9], [10]. In this work we focused on the more recent runtime called Swift/T (for the Turbine runtime [20]) geared toward supporting exascale workloads. The Turbine runtime implements a highly integrated workflow model that launches itself a single, large MPI application and optionally dedicates communicators to subtasks. This approach may also be especially interesting for applications considering use of in situ techniques [21].

Swift/K, on the other hand, provides a runtime similar to that of Cylc, adopting a model of job-based workflows potentially distributed across multiple sites. In the language shared by Swift/K and Swift/T, the data model can become very fine-grained, with data objects potentially as small as single integers. Each Swift variable is a future, and tasks are represented as Swift functions that block on their inputs.

b) Cylc: Cylc implements a distributed workflow model, with data generation and postprocessing and preprocessing potentially being executed on different sites and consisting of multiple jobs. Hence, Cylc also allows the collection of results from multiple compute sites. In Cylc, users define a directed dependency graph of tasks, as well as data objects that are passed between tasks. Cylc provides an easy mechanism to export a graphical representation of the workflow in dot format. Cylc is used mostly in the context of climate and weather applications.

D. Holistic I/O Monitoring

In this section we introduce four building blocks that assist in capturing a comprehensive picture of I/O activity throughout a data center. Darshan provides the means to capture I/O activity from an application's perspective. Many storage subsystems additionally provide their own, sometimes proprietary, mechanisms to query backend I/O monitoring data. TOKIO collects this data from Darshan and other relevant I/O subsystems and combines it for further analysis.

a) Darshan: Darshan captures I/O activity related to the application and I/O library layers in an HPC stack. In particular, Darshan can instrument many I/O interfaces (POSIX, STDIO) and libraries (such as MPI, HDF5) and can collect and aggregate its own I/O-related performance counters. Figure 2 illustrates at which levels Darshan is able to collect I/O activity within a system stack. Darshan with extended tracing support (DXT) [22] also enables the collection of full I/O traces up to a configurable size. In many cases, instrumenting an application is as easy as using `LD_PRELOAD` to interpose Darshan's instrumentation library between the application and its I/O libraries. This approach can be used by all users without requiring special privileges.

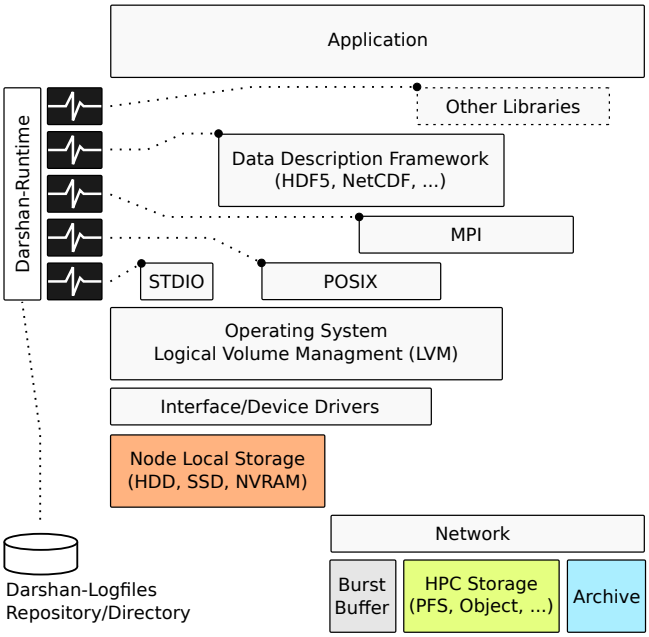


Fig. 2: With Darshan one can collect I/O-related activity on the application and library levels without requiring special privileges. The dotted lines to STDIO, POSIX, MPI, and HDF5 depict some instrumentation supported with Darshan by default, but users can define additional wrappers for other libraries as well. Recorded log data is stored into log files before a group of MPI processes terminates.

b) System-Level Monitoring/Vendor APIs: I/O monitoring data is generally not accessible by all users, and for proprietary systems internal performance counters may not be exposed even to site operators. Many modern systems, however, do offer APIs to query different system statistics or even provide event hooks for more sophisticated actions.

c) TOKIO: The TOKIO (Total Knowledge of I/O) framework [23] brings together monitoring information from multiple sources throughout the data center, as illustrated in Figure 3. This framework includes access to privileged monitoring information from I/O subsystems and vendor APIs, but it also makes use of application data collected by using Darshan. By continuously monitoring a data center over time, one can, for example, detect performance regressions [24]. TOKIO also offers tools to analyze and aggregate collected log records.

d) Telemetry Support in Workflow Engines: Collection or integration of telemetry information is also built into some workflow engines, although I/O performance is not addressed

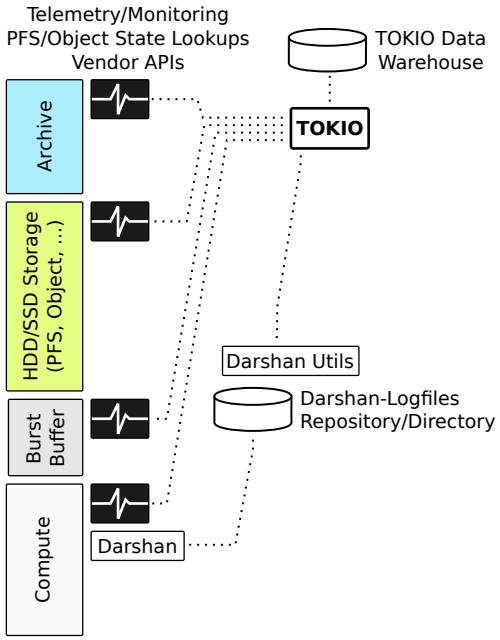


Fig. 3: TOKIO takes a holistic approach to I/O activity capture throughout the data center. To do so, TOKIO collects data from different data sources, such as system and service logs, vendor APIs, PFS monitoring tools, and Darshan log files.

specifically. The Tigres WMS [25], for example, features a monitoring API to log events. Tigres also was evaluated for use with I/O-intensive workflows, but a facility within the workflow engine to capture I/O behavior is missing. Cylc [11] monitors jobs and keeps track of output files even across multiple sites. Fireworks [13] has a tracking feature for files to check whether a file has enough output lines. Ophidea [17] captures walltime and other statistics for workflows. Swift/K allows for external monitoring via the web browser, a Java Swing tool, or an ANSI text user interface. Swift/T uses optional text or more scalable MPE logging [26].

External tools and dashboards have also been developed to monitor workflows or WMS. Grafana dashboards, for example, have been created [27] to monitor Spark.

III. Design Challenges

With the background on HPC workflows and I/O activity capture established, this section introduces design challenges for an architecture to gather, analyze, and present the I/O behavior of workflows. The main motivation for the tools introduced here is to gain insights useful for operating decisions and system design. Because addressing identified problems or optimization opportunities usually requires coordination with users and application developers, the tools also aim to provide more intuitive means of communication. We achieve this by associating I/O-related observations for different subsystems, which regular users may even be unaware of, with the processes that users can relate to since the processes are part of the workflows. The information collected by these tools is expected to become a valuable input source for the realization of smarter systems (see Section III-B). A potential integration with resource management and I/O-related middleware in the future is discussed in Section V-A and Section V-B.

To be useful in the HPC context, the architecture has to support multiple WMS because communities use different tools. The architecture also has to take into account that supercomputing sites can vary significantly, as evidenced in a variety of different scheduling systems, storage systems, and software environments with a wide range of versions for a specific software tool or library.

To address these requirements, we adopted a modular approach, with a first iteration of the prototype being designed with three user perspectives in mind:

1) I/O researchers: Expect a flexible environment to explore analysis data; for large amounts of log data, interactive elements will be helpful
2) Site operators: Expect a toolbox that can be customized to fit the needs of the data center or individual subsystems
3) Application scientists: Expect a report that can be related to the processing steps performed in their applications and workflows

A first architecture for comprehensive I/O analysis of HPC workflows is depicted in Figure 4. Workflows can be defined implicitly or explicitly. Finding implicit workflows, as far as I/O is concerned, presents researchers with challenges. It seems reasonable, however, that at least parts of workflows can be discovered from log data and I/O activity records that might be used to improve system performance.

For users who use workflow engines (e.g., Swift, Cylc) there exists an explicit description of the workflow, which usually will be a directed acyclic graph of tasks and data objects. A WMS should offer a easy mechanism to export the DAG for visualization and other tasks; and, in fact, some do provide mechanisms for exporting the DAG (e.g., in dot format). In any case, we assume that one can often obtain a dependency graph. We then assume helper utilities supporting different WMS that transform the dependency graph into a preliminary workflow report that can then be populated with I/O activity records and annotations.

```
{
 nodes: [{type: "task", ...}, {type: "file", ...}],
 edges: [...],       # tasks/data dependencies
 reports: {
  tasks: [...],     # task reports
  files: [...],     # file reports
 ... },
 annotations: [...] # e.g., advice
}
```

Fig. 5: Structure of a workflow report featuring (1) the workflow dependency graph of tasks, files, and edges for their relationships; (2) reports associated with different elements of the workflow; and (3) annotations and advice also for different elements of the workflow.

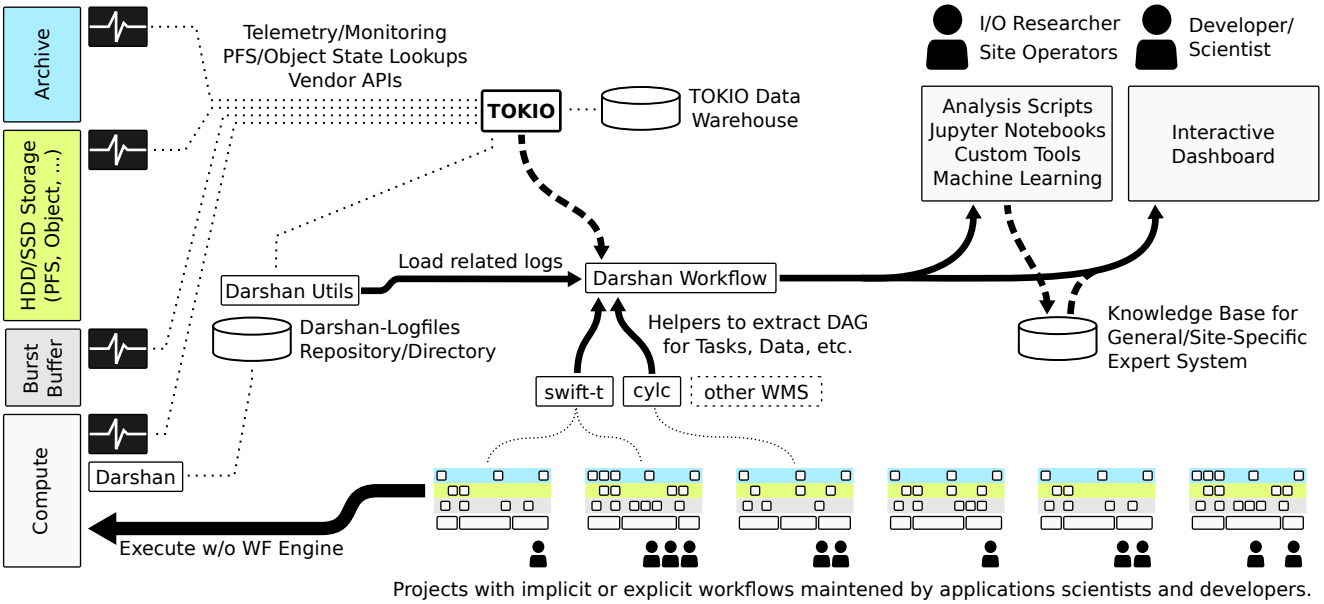


Fig. 4: Architecture overview to augment I/O behavior in HPC workflows by pulling in information from workflow engines and I/O activity capturing tools. Integration with TOKIO and expert systems is not yet implemented in the proof of concept introduced in Section IV as indicated by the dashed arrows. Thin dotted lines represent (optional) information sources.

The general structure of such a workflow report is illustrated in Figure 5. The report JSON provides expected fields for the workflow dependency graph (`nodes`, `edges`). The `nodes` field typically will hold tasks (e.g., jobs) or data objects (e.g., files). The `reports` field currently assumes referenced or inlined Darshan reports. Since Darshan reports can be broken up, for example into reports per file accessed (many different criteria can be used to aggregate the individual performance counters), the `reports` field also distinguishes between tasks and files.

For batch scheduling workflow engines, a task usually corresponds to a job on an HPC system, for which Darshan will record log files when an MPI application is invoked. The `annotations` field is used to allow experts to add advice either manually or, potentially, automatically using machine learning models.

A. Associating Tasks and Data Objects with Telemetry

This section covers some challenges related to the association of workflow tasks with recorded I/O activity. The first challenge is to find a suitable mapping between our abstraction for tasks and data objects and the notion used by the WMS. A second challenge is to identify and store cues about how tasks/data and log records relate to each other.

Unfortunately, mechanisms for automatic or transparent association are not widely supported by workflow engines and monitoring tools. Multiple mechanisms do exist, however, to perform logdata/telemetry association are possible. The following are two approaches that can be realized without requiring extensive changes to existing monitoring solutions and workflow engines.

- Naming conventions so that, for example, execution binaries and tasks match. Obviously, this is a fragile approach and may occasionally fail or pull in unrelated information.
- Exporting the name of the current task into an environment variable. This is the preferred method to work with Darshan, but it has limitations when working with workflows that execute multiple tasks within a single MPI execution.

For more integrated approaches such as followed by Swift/T (see Section II-C0a), logging granularity of MPI executions or at the job level is usually not meaningful anymore. In the case of Swift/T, for example, a single MPI-based runtime is started that uses MPI communicators to grant compute resources to tasks under the assumption that tasks are defined in a way that honors the provided communicator. Darshan at the moment captures this activity; however, it does not maintain individual performance counters per MPI communicator. As a result, the counters collected in log files are the superposition of many different tasks that cannot be easily broken down into individual tasks again. Maintaining per communicator counters is also not always desirable because it substantially increases the footprint of the instrumentation and log files. In addition, monitoring tools such as Darshan might need to provide an API to allow workflow engines to interact with the instrumentation, for example, to notify about the active context/task and to store some custom metadata to log records.

While this API requires changes to monitoring tools such as Darshan, another requirement for workflow engines is that they expose and therefore keep track of which task is active for a given process. In integrated approaches, a workflow engine may be in charge of notifying a monitoring solution of the context/task switch. Since this can be associated with

overheads for the WMS, the architecture cannot rely on this feature.

Another use of such an API may be to expose information about for what data objects and files are being used, which might be added to log records to improve analysis and visualization. Table I indicates whether the WMS assumes or implements a data model along with the workflow execution functionality. While some WMS such as Spark, Swift, or Ophidia use data models to optimize domain decomposition and minimize data movement, no workflow engine offers users the opportunity to add annotations on how the data is intended to be used.

Such a capability would allow filtering for which workflow elements logging data has to be gathered. Moreover, this could help apply different analysis depending on how a data object is used (also see Section III-B).

Scaling Considerations: A single workflow already might quickly accumulate millions of log records with varying degrees of detail and scattered across different subsystems. Challenging research questions are how to best address this explosion of log records and how to sample a representative subset of log data.

Darshan log files, for example, are usually stored to the parallel file system. For log data discovery this approach is not always ideal because it might require scanning the file system to discover logfiles. An alternative is to populate a separate index with references to log files as tasks are being executed, thus creating the association at runtime. Workflows that did not use a WMS and monitoring solution with support for this feature will still fall back to the first mode of log data discovery.

B. Integration of Expert Systems and Machine Learning

The proposed architecture explicitly assumes the integration with machine learning and big data analysis to generate insight as well as reacting to or displaying advice in an expert system scenario. While not implemented at this time, potential advisories an expert system could provide include the following:

- Detection of reliance on POSIX features such as timestamps, for example for locking/semaphore files or diagnostics. In addition, an alternative service or feature could be recommended if available.
- Detection and indication of tasks that are issuing large amounts of random I/O.
- Detection of inadequate Lustre/LFS stripe configurations as well as better recommendations.
- Advice on most suitable storage system from a list of available systems in a heterogeneous data center.
- Detection of inappropriate use of MPI collective/independent I/O as well as overlapping reads on the same file in a parallel application.
- Recommendation of the optimal checkpoint frequency based on observed reads/writes. (This requires explicit declaration or reliable classification as checkpoint file; compare Section III-A).
- Recommendation of HDF5 chunking options. Useful, for example, with write-once-read-many datasets and workflows where multiple follow-up tasks might access the data by using distinct access patterns.

Accommodating the previous use cases gives rise to a number of requirements for the workflow report structure and features to be supported by the workflow processing toolchain:

- Reserved entries, which are picked up by standard tools
- Custom (opaque) entries in workflow report
- Representations compatible to wide-spread data analysis and machine learning tools (e.g., compatible to numpy)
- Hooks for modules to specify processing actions and visual representation

Two scenarios which integrate advisory products derived from workflow I/O activity with scheduling systems and I/O middleware are discussed in Section V-A and Section V-B.

Machine learning techniques also offer potential to discover implicitly defined workflows by analyzing which jobs and users have been accessing which files. For workflows that are being executed sufficiently often, by using pattern recognition and clustering methods this capability might be possible even when data object identifiers and file names are varying. This would be of direct help for users, but it could also be interesting for automated systems and adaptive scenarios such as those discussed in Section V.

IV. Proof of Concept Implementation

This section offers a description and evaluation of the first prototype implementation of our proposed approach. There are multiple approaches for a proof of concept implementation which complement each other, yet, each in itself already offers opportunities to generate insight:

- Add fine-grained logging capabilities to Darshan (in particular, per MPI communicator accounting)
- Expose an API for the workflow engine to inform Darshan of task boundaries and context switches
- Focus on cases where tasks generate distinct darshan logs

The first two approaches require delicate changes and feature additions touching the core architecture of Darshan and workflow engines. In particular, these might lead to the introduction of performance penalties or increased memory footprints. albeit the work presented here does not. Our approach, focusing on existing logging capabilities, is most applicable to traditional HPC applications submitted to batch scheduling systems. The proof of concept therefor supports use cases for two different workflow engines to begin with: Swift/T (see Section II-C0a) and Cylc (see Section II-C0b).

After a brief introduction on how workflows are declared and extracted from different WMS in Section IV-A, Section IV-B outlines the basic steps to compile a workflow report. The remainder of this section covers tools to visualize and explore I/O for workflows interactively (see IV-C) as well as in scripts (see IV-D). Finally, we list implications and recommendations for third parties to help compile an I/O overview for workflows in Section IV-F.

A. Example Workflow and Workflow Declaration

To demonstrate how the approach helps to gain insight and how the tools work, we prepared a simulation workflow similar to the one introduced in Section II-B, as it might be performed by a climate modeler. The workflow was kept simple for illustrative purposes and for easy comparison between subsections. The demonstration workflow consists of the following sequence of tasks:

1) Preprocessing: Multiple input files are combined to create the initial state for a simulation.
2) Simulation: Multiple timesteps are simulated, and a timestep file is written that is read by the next simulation step as the initial state.
3) Postprocessing: For every simulated timestep some post-processing is performed. In this example, the results of different tasks are appended to a file shared by multiple tasks (imagine a timeseries or a movie).

Depending on the WMS, the description of a workflow can vary considerably. Compare, for example, the workflow definition used by Cylc in Figure 6 with the workflow definition as it could be defined using Swift in Figure 7.

In Cylc, a user defines tasks, in this case, for example, as a command line command to be executed in the `runtime` section. In a separate section the dependencies are defined, including convenience features such as the so-called cycle features that can spawn a pipeline of events for different input parameters. The application logic and workflow description and configuration are neatly separated.

```
[scheduling]
 initial cycle point = 2021
 final cycle point = 2023
 [[dependencies]]
  [[[R1]]]  # Initial cycle point.
   graph = prep => model
  [[[R//P1Y]]]  # Yearly cycling.
   graph = model[-P1D] => model => post
  [[[R1/P0Y]]]  # Final cycle point.
   graph = post => stop

[runtime]
 [[prep]]
  script = mpiexec -np 1 ./prep
 [[model]]
  script = mpiexec -np 4 ./model
 [[post]]
  script = mpiexec -np 1 ./post
```

Fig. 6: Example of a workflow defined using Cylc.

A Swift workflow, on the other hand, looks more like a normal programming language. As a result, identifying individual tasks is not as straightforward. At this time, association with Swift tasks relies on a convention to have function calls in Swift coincide with the names of binaries to be executed in parallel. This choice was made in order to limit the granularity of the dependency graph, since Swift keeps track of individual variable instances and scopes. Instead of dividing aggregate I/O activity for tasks, a more appropriate approach is to associate records with source lines or elements of the acyclic syntax tree of the Swift script. Using the Swift compiler (stc), a Tcl script is generated that can be executed using the *turbine* runtime.

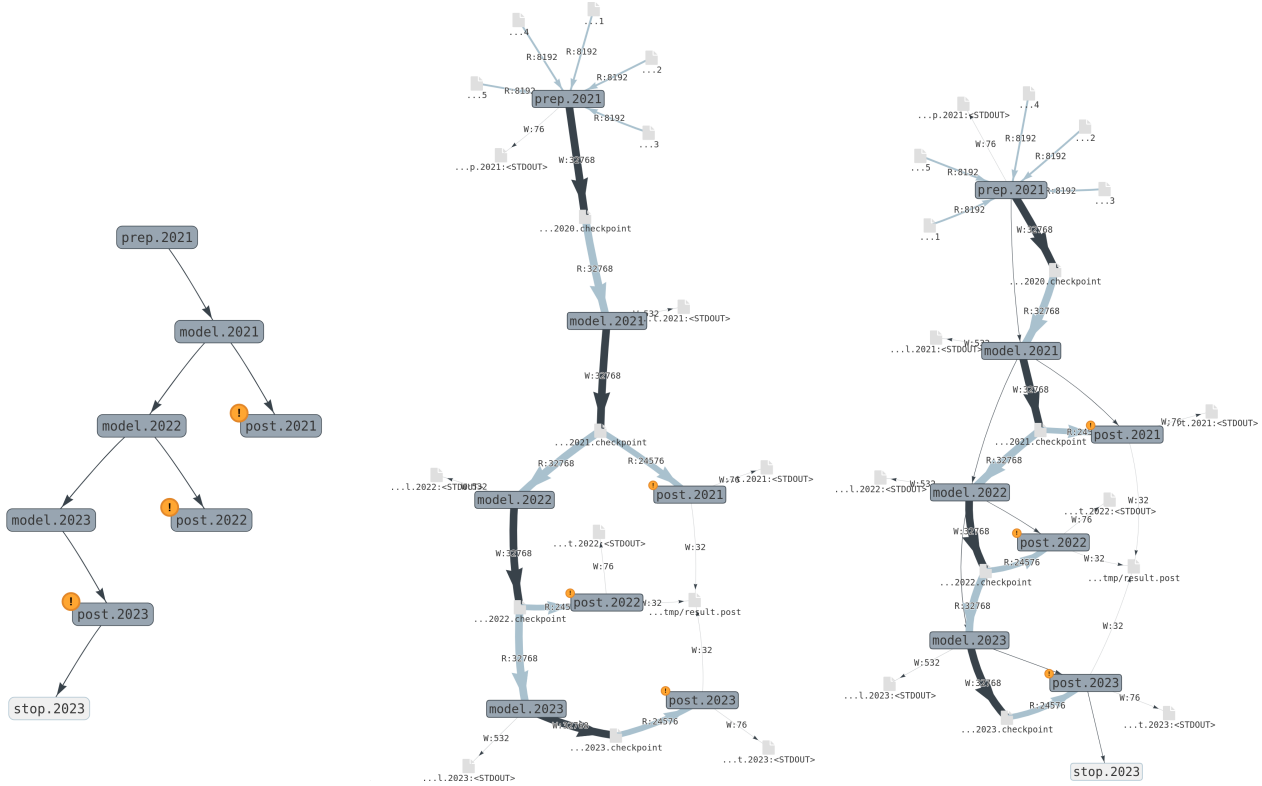
```
int m[]; // assuming integer references to datasets
int p[];

m[2020] = prep();       // a pre-processing step

foreach x in [2021:2023] {
  m[x] = model(m[x-1]);   // a simulation
  p[x] = post(m[x]);   // a post-processing step
}
```

Fig. 7: A simple workflow defined in Swift using pseudo code for the range and array semantics for brevity.

B. Workflow Report Generation

This section describes the different processing steps necessary to generate the workflow overview report (see Figure 5).

1) Provided a workflow was defined using a WMS (at the moment either Swift or Cylc).
   a) Enable instrumentation with Darshan (this can usually be achieved by setting and exporting `LD_PRELOAD`).
   b) Configure the workflow engine to expose a TaskID within the execution environment. A modified variant of Darshan uses this information to add metadata to Darshan logs or to use in the log file name.
2) Extract a dependency graph of the workflow. This could be either derived from source, via an export feature to dot, or generated from an execution log. A helper script for the workflow engine maps the WMS task/data model to the task/data abstraction used by *darshan-workflow* postprocessing tools.
3) Using the nodes in the workflow graph, discover Darshan logs for every task (currently by scanning the file system). Ideally, the lookup could also be done from a database or index files which were populated directly by Darshan or the WMS. Note that, currently, Darshan logs can only be generated by MPI applications (specifically, Darshan relies on the application to call MPI_Init and MPI_Finalize).
4) Preprocess Darshan logs, and convert them into JSON representation or Python data objects for further analysis. Preprocessing will typically generate a number of derived performance counters (per file, per operation).
5) Generate a workflow report file, including data/references for preprocessed reports.
6) (Optional) Add additional advisory products manually by review (human in the loop) or by using machine learning for automatic analysis (can be already added and rendered with interactive tools).
7) Interact with data using Python/Jupyter Notebooks or the report dashboard example.



(a) Dependency view as defined by using a workflow management system or seen by the user.

(b) Dataflow view with files and the number of total bytes read/written per task/file added to the graph.

(c) Hybrid view

Fig. 8: Different perspectives for visualization of the same workflow.

C. Visualization and Interaction

The large number of tasks and log records, as discussed in Section III-A, gives rise to tools allowing for convenient exploration of workflow I/O activity. As outlined in Section II, graph-based representations for workflows are popular among a number of WMS, but an I/O perspective is usually not offered. For visualization tools, users may have different preferences and needs, including the following:

- Users may prefer to integrate I/O feedback with any GUI tools their workflow engines supports. Whether this is feasible from a software support perspective is unclear.
- Users and operators may use well-established dashboards such as Grafana or Nagios. Instead of requiring yet another tool, many ecosystems offer mechanisms to extend these systems with custom widgets.
- Sometimes specialized or custom tools may be most appropriate. E.g., to best communicate the I/O perspective, reports may require visualization, plots, and interface elements optimized for this perspective.

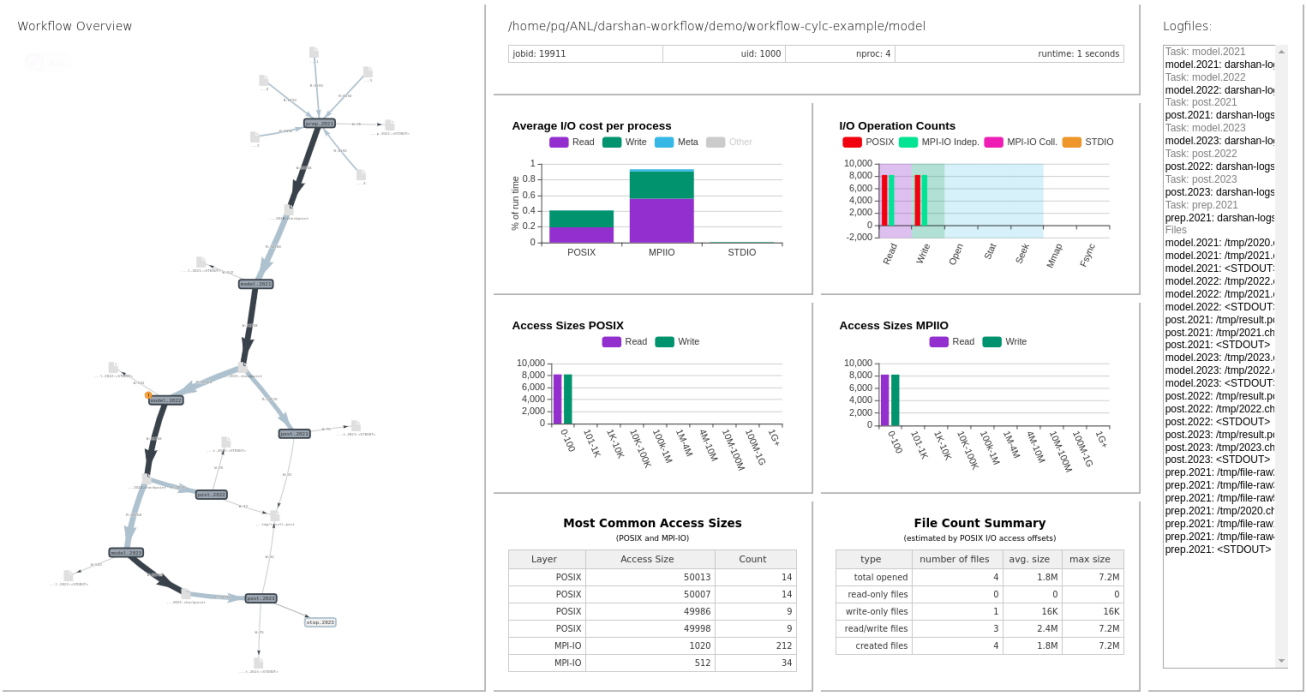
Using vis.js [28] as a foundation, we implemented graph representations for different perspectives and different levels of detail, in order to render workflows using the format shown in Figure 5. Javascript packaging allows for easy reuse in various contexts and custom tools such as Jupyter Notebooks (see Section IV-D) and dashboards (see Section IV-E).

Figure 8a illustrates a dependency view, resembling the graph representation most WMS would offer to users. In Figure 8b a dataflow or I/O activity perspective is rendered that uses tasks from the workflow description and adds files and read/write access information obtained from Darshan log records. The edge label and the width of an edge indicate the number of total bytes written in absolute and relative terms. Mixing both perspectives as shown in Figure 8c can be useful, although challenges exist in automatically choosing which information to display.

We also note the orange exclamation marks rendered near the top left corner of the postprocessing tasks. Indicators like this (e.g., to indicate usage of MPIIO, POSIX, or STDIO) can be used to communicate problems and other information in a compact and intuitive representation. Currently, an indicator is added when an annotation for a node/task is defined in the JSON report.

D. Libraries for Use in Custom Tools and Jupyter Notebooks

Tool development in research is a delicate task since integrated solutions quickly cease to address unforeseen use cases. Instead of aiming for such an integrated solution, the toolchain to extract the workflow description, gather Darshan/TOKIO log records, and process and aggregate the log data was



Layer	Access Size	Count
POSIX	50013	14
POSIX	50007	14
POSIX	49986	9
POSIX	49998	9
MPI-IO	1020	212
MPI-IO	512	34

type	number of files	avg. size	max size
total opened	4	1.8M	7.2M
read-only files	0	0	0
write-only files	1	16K	16K
read/write files	3	2.4M	7.2M
created files	4	1.8M	7.2M

Fig. 9: Executing a workflow may produce substantial amounts of log data. Interactive tools such as dashboards can help explore workflows in detail. The graph shows a dataflow view with edges of workflow dependencies removed.

chosen to use Python/Javascript libraries where expedient. This limits code duplication across specific tools as used in glue code and helpers, but it also allows the user to quickly adapt analysis or integrate with other Python-based libraries. This is of particular interest since the ecosystem for data science, machine learning, and neural networks is well developed for Python (e.g., Keras, Tensorflow, Pandas, scikit-learn). While aiming for smaller reusable components may add initial overhead, the effort pays off in the long run as it offers flexibility.

As an example, the workflow visualizations described in Section IV-C are compatible with Jupyter Notebooks by providing a special Jupyter Widget. The motivation for support in this direction is to endorse rapid prototyping and reproducible I/O analysis for workflows. Jupyter Notebooks, for example, can be easily shared.

E. Workflow I/O Dashboard for Users

The benefits of considering visual and interactive tools were already motivated in Section IV-C and Section IV-D on visualization and modular packaging. In this section we showcase a functioning special-purpose dashboard to interactively explore and visualize the I/O activity related to workflows. A screenshot of the dashboard is shown in Figure 9. Users can click on and select individual or multiple nodes in the workflow graph or in the report sidebar. As the selection changes, the report summary and the plots in the middle section are updated. Where appropriate, performance counters of multiple reports can be aggregated and displayed in a combined statistic. Besides the interactive elements, providing such a dashboard to users is attractive because it allows the inclusion of activity data that otherwise is not available to users, since the interface to query log records in many cases requires special privileges. Should the log data contain sensitive information, processes can be established to audit and remove sensitive information. In other cases, after aggregation or after plotting to static images, no sensitive information remains.

F. Implications for WMS, Monitoring, and Developers

Development of the prototype implementation helped identify a number of ways that WMS, monitoring solutions, and application developers can help make collection and association of log records easier. For WMS the recommendations are as follows:

- Expose active task and offer facilities for backtrace.
- Export workflow dependency graph, for example, in dot format.
- Use data/file notions (e.g., to declare data object is snapshot, diagnostic, dataset)

For monitoring tools on the application and library layer the following features would be helpful:

- Provide options to pick up context to allow associations.
- Support user-specific metadata with records.
- Offer an API to interact with monitoring toolkit.
- Allow performance counters per MPI communicator.

Application developers can help by making their intent more explicit by using libraries (e.g. HDF5) or domain-specific languages. Scientists running workflows should consider enabling tools such as Darshan with at least a subset of their runs to help create a body of training data for adaptive systems.

V. Use Cases

This section outlines two use cases that become possible as a result of better insight into HPC workflows in a mid- to long-term perspective. Section V-A outlines a scenario where knowledge about the I/O phases of workflows and tasks might allow more efficient I/O-aware scheduling. Section V-B illustrates a pathway for I/O-related middleware to make data placement decisions to conserve cost or to optimize performance.

A. Use Case: I/O Aware Scheduling

In a scheduling scenario, knowledge about the I/O phases of a task, job, or pipeline offers interesting opportunities to adapt scheduling decisions. Figure 10 illustrates an example that assumes a number of independent jobs that are submitted to a batch scheduling system. Assuming a workflow that experiences bursty I/O, this approach could help the scheduler spread jobs over time in order to reduce stress on the file system. To do so, the scheduler could ensure that different jobs are started out of phase in order to spread out I/O more evenly over time. This approach could allow operators to offer more predictable quality of service or to reduce cost by scaling down system size since peak performance requirements could be relaxed.

B. Use Case: Decisions in I/O Middleware

A second use case concerns I/O-related middleware such as HDF5 [29], Decaf [30], ADIOS [31], and ESDM [32]. Across the I/O stack countless decision components are active. By considering knowledge about the different processing steps of long-living data in workflows, a number of decisions could be improved:

- Domain decomposition on write
- Target (service, client/server, OST) selection
- Logical separation of data (e.g., metadata vs data), and autopopulating of data catalogue services

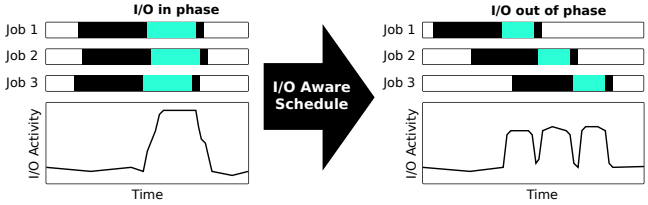


Fig. 10: Sketch of I/O-aware scheduling using knowledge about workflows to potentially increase performance and reduce I/O time per job or allow procuring a scaled-down storage system.

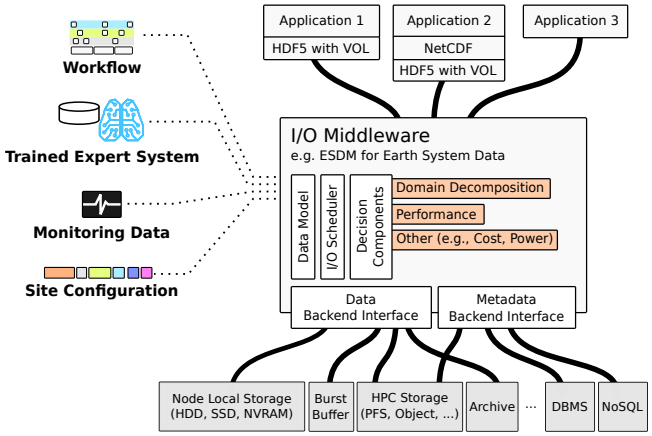


Fig. 11: I/O middleware considering knowledge about workflow I/O to make data placement or transformation (e.g., enabling compression) decisions.

In particular, applications with high degrees of similarity and minor variation in the structure of output data are attractive for this approach. For UQ and HTC workflows, for example, a pipeline may be executed hundreds or thousands of times. Because doing so yields multiple observations per task and pipeline, we can be more confident in the expected behavior and slowly and potentially automatically adapt the system behavior to better accommodate this workload. The I/O middleware might need to consider information from multiple sources, as is illustrated in Figure 11. This includes the I/O perspective on workflows, knowledge from expert systems about tunables and optimal configurations, feedback with current monitoring data, and the available systems and service of the data center.

VI. Summary and Conclusion

Capturing a holistic picture of I/O activity for workflows is an ambitious goal, in part because of the fragmented landscape of tools for workflow description as well the variety of monitoring records used across I/O subsystems. In this work we demonstrate that it is possible to generate useful reports and interactive visualizations with reasonable effort in two WMS's used by scientists for production applications. The underlying methodology can be extended and applied in a variety of other workflow environments as well.

The current state of the practice in monitoring workflow I/O activity starts with an existing workflow definition and I/O activity collected on the application or library layer. The proposed architecture, however, can incorporate log records from across a data center. In particular, the approach addresses the diversity of stakeholders by adopting a modular architecture that provides a common core library to provide functionality that is then reused in glue code, special-purpose tools, and for integration with third-party tools.

Besides low-level integration with WMS and the collection of record data to generate an overview report, we showcased a number of ways to use workflow reports more effectively.

In particular, this objective is achieved by providing widgets and a Python library to support common tools such as Jupyter Notebooks to encourage reproducible I/O analysis of workflows. Furthermore, we discussed the use of interactive tools and dashboards to provide intuitive ways for users and I/O researchers to explore workflows.

The implementation of the proof of concept also revealed a number of ways (see Section IV-F) that existing WMS and monitoring solutions as well as application developers and scientists can help better understand HPC workflows.

In the future, we plan to apply the existing toolchain to analyze additional practical workflows and address anticipated scaling problems. In particular, we welcome suggestions from the community about data-intensive workflow candidates with an exascale use case in mind. We also would like to add support to additional WMS, improve the support utilities and libraries, and try the tools on more sites.

ACKNOWLEDGMENT

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

The ESiWACE project received funding from the EU Horizon 2020 research and innovation programme under grant agreement No 675191.

REFERENCES

[1] Top500, "Top500 Supercomputer Sites." [Online]. Available: http://www.top500.org/

[2] DOE and NISA, "Exascale Computing Project (ECP)," 2017. [Online]. Available: https://www.exascaleproject.org/

[3] Intel, The HDF Group, EMC, and Cray, "Fast Forward Storage and I/O," Jun. 2014.

[4] "NEXTGenIO: Next Generation I/O for the Exascale." [Online]. Available: http://www.nextgenio.eu/

[5] G. K. Lockwood, D. Hazen, Q. Koziol, S. Canon, K. Antypas, J. Balewski, N. Balthaser, W. Bhimji, J. Botts, J. Broughton, T. L. Butler, and G. F. Butler, "A Vision for the Future of HPC Storage," p. 37, Oct. 2017.

[6] G. K. Lockwood, W. Yoo, S. Byna, N. J. Wright, S. Snyder, K. Harms, Z. Nault, and P. Carns, "UMAMI: A Recipe for Generating Meaningful Metrics Through Holistic I/O Performance Analysis," in *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, ser. PDSW-DISCS '17. New York, NY, USA: ACM, 2017, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/3149393.3149395

[7] LANL, NERSC, and SNL, "APEX Workflows," Mar. 2016. [Online]. Available: https://www.nersc.gov/assets/apex-workflows-v2.pdf

[8] 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, Jan. 2018. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1094342017704893

[9] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/T: Scalable data flow programming for distributed-memory task-parallel applications," in *Proc. CCGrid*, 2013.

[10] T. G. Armstrong, J. M. Wozniak, M. Wilde, and I. T. Foster, "Compiler techniques for massively scalable implicit task parallelism," in *Proc. SC*, 2014.

[11] "Cylc -A Workflow Engine." [Online]. Available: https://cylc.github.io/cylc/

[12] "RDD Lineage — Logical Execution Plan Mastering Apache Spark." [Online]. Available: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-lineage.html

[13] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, and K. A. Persson, "FireWorks: A Dynamic Workflow System Designed for High-Throughput Applications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5037–5059, 2015, cPE-14-0307.R2. [Online]. Available: http://dx.doi.org/10.1002/cpe.3505

[14] 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, May 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167739X14002015

[15] NERSC, "TaskFarmer," 2017. [Online]. Available: http://www.nersc.gov/users/data-analytics/workflow-tools/taskfarmer/

[16] "Tigres." [Online]. Available: http://tigres.lbl.gov/home

[17] C. Palazzo, A. Mariello, S. Fiore, A. D'Anca, D. Elia, D. N. Williams, and G. Aloisio, "A Workflow-Enabled Big Data Analytics Software Stack for Escience," in *2015 International Conference on High Performance Computing Simulation (HPCS)*, Jul. 2015, pp. 545–552.

[18] "The Kepler Project — Kepler." [Online]. Available: https://kepler-project.org/

[19] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[20] J. M. Wozniak, T. G. Armstrong, K. Maheshwari, E. L. Lusk, D. S. Katz, M. Wilde, and I. T. Foster, "Turbine: A distributed-memory dataflow engine for high performance many-task applications," *Fundamenta Informaticae*, vol. 28, no. 3, 2013.

[21] M. Dorier, M. Dreher, T. Peterka, J. M. Wozniak, G. Antoniu, and B. Raffin, "Lessons learned from building in situ coupling frameworks," in *Proc. In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization @ SC*, 2015.

[22] C. Xu, S. Snyder, O. Kulkarni, V. Venkatesan, P. Carns, S. Byna, R. Sisneros, and K. Chadalavada, "DXT: Darshan eXtended Tracing," p. 8, 2017.

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

[24] G. K. Lockwood, T. Wang, S. Byna, N. J. Wright, S. Snyder, and P. Carns, "A Year in the Life of a Parallel File System," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC18)*, 2018.

[25] V. Hendrix, J. Fox, D. Ghoshal, and L. Ramakrishnan, "Tigres Workflow Library: Supporting Scientific Pipelines on HPC Systems," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 146–155.

[26] J. M. Wozniak, A. Chan, T. G. Armstrong, M. Wilde, E. Lusk, and I. T. Foster, "A model for tracing and debugging large-scale task-parallel programs with MPE," in *Proc. Workshop on Leveraging Abstractions and Semantics in High-Performance Computing (LASH-C) at PPoPP*, 2013.

[27] "Grafana-Spark-Dashboards: Scripts for Generating Grafana Dashboards for Monitoring Spark Jobs," Hammer Lab, Aug. 2018. [Online]. Available: https://github.com/hammerlab/grafana-spark-dashboards

[28] Almende B.V., "Vis.Js - A Dynamic, Browser Based Visualization Library." 2018. [Online]. Available: http://visjs.org/

[29] "HDF5: Hierarchical Data Format." [Online]. Available: https://www.hdfgroup.org/hdf5/

[30] T. Peterka, Franck Cappello, and Jay Lofstead, "Decaf: High-Performance Decoupling of Tightly Coupled Flows — Argonne National Laboratory," 2018. [Online]. Available: http://www.mcs.anl.gov/project/decaf-high-performance-decoupling-tightly-coupled-flows

[31] "Next generation of ADIOS developed in the Exascale Computing Program: Ornladios/ADIOS2," ADIOS, Sep. 2018. [Online]. Available: https://github.com/ornladios/ADIOS2

[32] "ESiWACE: Centre of Excellence in Simulation of Weather and Climate in Europe." [Online]. Available: https://www.esiwace.eu/