End-to-End Online Performance Data Capture and Analysis for Scientific Workflows

George Papadimitriou[a,*], Cong Wang[b], Karan Vahi[a], Rafael Ferreira da Silva[a], Anirban Mandal[b], Zhengchun Liu[c,d], Rajiv Mayani[a], Mats Rynge[a], Mariam Kiran[e], Vickie E. Lynch[f], Rajkumar Kettimuthu[c,d], Ewa Deelman[a], Jeffrey S. Vetter[f], Ian Foster[c,d]

[a] *University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA*
[b] *RENCI, University of North Carolina Chapel Hill, NC, USA*
[c] *Data Science and Learning Division, Argonne National Laboratory, IL, USA*
[d] *University of Chicago, Chicago, IL, USA*
[e] *Energy Sciences Network (ESnet), Lawrence Berkeley National Labs, CA, USA*
[f] *Oak Ridge National Laboratory, TN, USA*

Abstract

With the increased prevalence of employing workflows for scientific computing and a push towards exascale computing, it has become paramount that we are able to analyze characteristics of scientific applications to better understand their impact on the underlying infrastructure and vice-versa. Such analysis can help drive the design, development, and optimization of these next generation systems and solutions. In this paper, we present the architecture, integrated with existing well-established and newly developed tools, to collect online performance statistics of workflow executions from various, heterogeneous sources and publish them in a distributed database (Elasticsearch). Using this architecture, we are able to correlate online workflow performance data, with data from the underlying infrastructure, and present them in a useful and intuitive way via an online dashboard. We have validated our approach by executing two classes of real-world workflows, both under normal and anomalous conditions. The first is an I/O-intensive genome analysis workflow; the second, a CPU- and memory-intensive material science workflow. Based on the data collected in Elasticsearch, we are able to demonstrate that we can correctly identify anomalies that we injected. The resulting end-to-end data collection of workflow performance data is an important resource of training data for automated machine learning analysis.

Keywords: scientific workflows, online performance monitoring, extreme scale.

1. Introduction

Automating the execution of computational tasks is necessary for improving scientific productivity. Scientific workflows have facilitated breakthroughs in several domains such as astronomy, physics, climate science, earthquake science, biology, among many others [1]. Large-scale workflows are typically comprised of thousands of tasks and process vast amounts of data (from remote sensors, instruments, etc.) to conduct complex modeling, simulations, and data analytics on distributed, heterogeneous resources. Scientific workflow management systems, such as Pegasus [2], are critical automation components that enable efficient and resilient workflow execution across heterogeneous infrastructures. As the workflow requirements keep increasing both in resource demands and complexity, there is an urgent need to understand the requirements and characteristics of these applications to drive the design, development, and optimization of the next-generation systems and solutions.

In spite of significant efforts on solving critical engineering challenges in workflow management [2] and the wide use of workflow systems executing many scientific applications from diverse domains in production [1], the current state-of-the-art approaches lack a deep understanding of the requirements, characteristics, and relationships of current and emerging applications and systems. As part of the DOE ASCR Next-Generation Networks for Science (NGNS) program [3], the Panorama [4] and RAMSES [5] projects have individually focused on the development of technologies and mechanisms to deepen our understanding of the applications and the systems. Panorama targets the collection, analysis, and sharing of performance data about end-to-end scientific workflows. RAMSES, on the other hand, targets the development of end-to-end analytical performance models that improve our understanding of the behavior of science workflows in extreme-scale science environments. By combining the strengths and solutions of data gathering at both the application and infrastructure levels jointly provided by Panorama and RAMSES, we aim to provide a complete end-to-end online system that seamlessly captures workflow characteristics and performance data of scientific workflows at runtime. The various outcomes of this work can be

*Corresponding address: USC Information Sciences Institute, 4676 Admiralty Way Suite 1001, Marina del Rey, CA, USA, 90292

Email addresses: georgpap@isi.edu (George Papadimitriou), cwang@renci.org (Cong Wang), vahi@isi.edu (Karan Vahi), rafsilva@isi.edu (Rafael Ferreira da Silva), anirban@renci.org (Anirban Mandal), zhengchun.liu@anl.gov (Zhengchun Liu), mayani@isi.edu (Rajiv Mayani), rynge@isi.edu (Mats Rynge), mkiran@es.net (Mariam Kiran), lynchve@ornl.gov (Vickie E. Lynch), kettimut@anl.gov (Rajkumar Kettimuthu), deelman@isi.edu (Ewa Deelman), vetter@ornl.gov (Jeffrey S. Vetter), foster@cs.uchicago.edu (Ian Foster)

applied in both theory and practice by improving application performance and driving the development of novel application-aware schedulers for workflow systems and by detecting and handling anomalies during workflow executions.

In this paper, we present how a collection of well-established tools, which have empowered computational science for the past decade, can be orchestrated to produce a comprehensive knowledge base of applications, systems characteristics, and requirements. Due to the complexity of scientific applications and the nature of distributed computing, building such knowledge not only requires information about the workflow application execution and its execution host, but also requires a broader understanding of the system as a whole. This includes capturing data at the application-level, both characteristics (e.g., workflow structure, input data, etc.) and performance (e.g., CPU and memory usage, I/O operations, etc.); and also at the system level within and across computing sites (e.g., bulk data transfers, networking including routing, IPv6, flow dynamics, etc.). Gathering, storing, and distributing such information is not only challenging due to the large volume of data produced by each system at different levels of the software stack, but also because of the need to properly identify common, overlapped, or complementary information generated by different tools. The system proposed in this paper not only automates data gathering, preparation, and storage, but also enables data querying, retrieval, and analytics. We leverage the scalable and flexible ELK stack (Elasticsearch, Logstash, and Kibana) [6], which enables automated and efficient data ingestion, discovery, retrieval, and visualization.

The main contributions of this paper include:

1. An end-to-end framework for online performance data capturing, preprocessing, and storing of scientific workflows;
2. A comprehensive set of real world workflow execution performance data, which includes fine-grained and coarse-grained application-level and resource-level data within or across computing sites;
3. Exemplar use of state-of-the-art frameworks and tools for enabling seamless integration, data capture, and improved understanding of current and next-generation applications and systems;
4. Exploration and query mechanisms for knowledge discovery via the ELK software stack, including a novel customized component for near-real time performance data visualization.

This paper is structured as follows. Section 2 presents an overview of the related work. Section 3 briefly describes the set of tools enabling this work. Section 4 presents the architecture of our end-to-end system for online data capture and analysis for scientific workflows, including our approaches to tackle data acquisition challenges. In Section 5, we present how this architecture can be deployed. Section 6 presents case studies with two real world scientific applications and experiments conducted to validate our approach. Section 7 provides information about our initial attempt to provide an open access data repository for workflow performance statistics. Finally, Section 8 concludes with a brief summary of results and a discussion of future research directions.

2. Related Work

The gathering and characterization of accurate resource usage is crucial for the development of solutions that enhance scientific productivity (such as tuned infrastructure configurations, and sophisticated resource allocation and task scheduling algorithms). As scientific applications and systems become more complex, understanding a system's behavior and its environment are key to efficient resource and application management. Although workload gathering and characterization is not novel, the fast pace at which such systems evolve requires new tools and mechanisms to efficiently process the large, heterogeneous volumes of data that they generate [7].

In the past decade, workload archives have become very popular [8, 9, 10] and have enabled several advances in distributed computing including the design, development, and evaluation of a number of algorithms [8]. Nevertheless, most of the algorithms are impractical due to the unrealistic assumptions, lack of meaningful comparison, or bindings to a specific platform. Additionally, such archives are not suitable for investigation of workflow-based applications as the dependencies between workflow tasks affect each other in terms of resource needs, performance, failures, and so on.

Some efforts have been made to collect, profile, and publish traces and performance statistics for real scientific workflows [11, 12, 13, 14, 15, 16, 17, 18, 19]. These traces provide fine-grained information about workflow and job characteristics and performance metrics including CPU and memory usage, I/O operations, job dependencies, among others. Although detailed workflow information could be extracted from these traces, there is limited information about the infrastructure as well as limited information about end-to-end performance data (e.g., TCP flows for data transfers, I/O profiling, etc.). In [11], we have profiled time-series data for the SNS workflow, in which anomalous behaviors could be identified. However, that work was limited to information gathered by the *pegasus-kickstart* toolkit. A common element absent from all past works is the lack of a system that automates data gathering and enables near real-time monitoring and investigation. The work described in this paper alleviates the absence of such an element by leveraging state-of-the-art tools and systems to provide a single entry point for data analysis of end-to-end, comprehensive, distributed workflow executions.

3. Software Ecosystem

Building a system for the end-to-end performance data capture of scientific workflow executions requires combining a collection of complex and heterogeneous tools. In this section, we briefly introduce the existing (and proven) software tools we have leveraged for each component of our proposed system architecture (presented in Section 4).

3.1. System Frameworks

Pegasus WMS. Pegasus [2] is a popular workflow management system that enables users to design workflows at a high-level of abstraction that are independent of the resources available to execute them and are also independent of the location of data and executables. Pegasus transforms these abstract workflows into executable workflows that can be deployed onto distributed and high-performance computing resources such as DOE Leadership Computing Facilities (e.g., NERSC [20] and OLCF [21]), shared computing resources (e.g., XSEDE [22] and OSG [23]), local clusters, and clouds. During the compilation process, Pegasus performs data discovery, locating input data files and executables. Data transfer tasks are automatically added to the executable workflow and perform two key functions: (1) stage in input files to staging areas associated with the computing resources, and (2) transfer the generated outputs back to a user-specified location. Additionally, data cleanup (removes data that is no longer required) and data registration tasks (catalog the output files) are also automatically added to the workflow. To manage user's data, Pegasus interfaces with a wide variety of backend storage systems that use different data access and transfer protocols.

During workflow execution, provenance information from workflow and job logs is automatically parsed and stored in a relational datastore by a monitoring daemon called *pegasus-monitord* [24]. Using a suite of associated command line tools and a web-based dashboard, users are able to monitor and debug their computations and determine a number of performance metrics about their workflows, including:

- *Workflow walltime* – the wall time from the start until the end of the workflow execution, which is reported by HTCondor DAGMan (the workflow executor used in Pegasus);
- *Workflow cumulative job wall time* – aggregated run times for all the individual jobs in the workflow (aids resource requirements estimation);
- *Breakdown of jobs by count and runtime* – for each job type, the total number of jobs (succeeded and failed), as well as the total, minimum, maximum, and average runtimes; and
- *Breakdown of tasks and jobs over time on hosts* – the number of jobs and total runtime of jobs running over different hosts.

The monitoring daemon can also be configured to send normalized events to an Advanced Message Queuing Protocol (AMQP) endpoint [25]. This is particularly useful when conducting analysis across workflows and correlating monitoring information from various monitoring sources.

Globus. The Globus transfer service [26] is a cloud-hosted software-as-a-service implementation that orchestrates file transfers between pairs of storage systems [27, 28]. A transfer request specifies, among other things, a source and destination; the file(s) and/or directory(ies) to be transferred; and (optionally) whether to perform integrity checking (enabled by default) and/or encrypt the data (disabled by default). Globus provides automatic fault recovery and automatic tuning of optimization parameters to achieve high performance. It can also transfer data using either the GridFTP or HTTP protocols. During the transfer, Globus provides performance monitoring metrics such as the average throughput at 60 seconds interval. Upon completion, a detailed transfer log is made available for the users, which includes:

- The request and completion time of the transfer;
- The source and destination endpoint information such as the number of physical Data Nodes (DTNs) and the type of transfer software stack (Globus Connect Personal or Globus Connect Server [29]);
- Transfer performance data such as the average transfer throughput, the total number of faults and checksum failures;
- Transfer parameters such as concurrency, parallelism, pipeline depth [30, 29], data integrity checking and encryption setting; and
- Transferred dataset information like total number of files, directories and bytes.

ELK Stack. The ELK stack consists of the Elasticsearch, Logstash, and Kibana [6] open-source tools. Elasticsearch provides a RESTful search and analytics endpoint. Logstash is a data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana lets users query and visualize data with charts and graphs in Elasticsearch. Combined, these tools provide a complete platform for data storage, retrieval, sorting, and analysis.

3.2. Monitoring Tools

Darshan. I/O performance behavior is gathered with Darshan [31], an HPC, lightweight, application-level I/O profiling tool that captures statistics about the behavior of HPC I/O operations. Darshan captures data for each file opened by the application, including I/O operation counts, common I/O access sizes, cumulative timers, and so on. I/O behavior is captured for POSIX IO, MPI-IO, HDF5, and Parallel netCDF data interface layers [32, 33, 34]. Darshan also captures a set of job-level characteristics such as the number of application processes, the job's start and end times, and the job unique identification provided by the scheduler. Lastly, Darshan can instrument I/O functions in both statically and dynamically linked executables.

Tstat. The TCP STatistic and Analysis Tool (Tstat) [35] is an open source, passive trace collection tool used to capture a packet-level log of a comprehensive set of parameters such as TCP's RTT, congestion window size, ACK/SYN/FIN counts, and retransmitted, reordered or lost packets. It also covers a wide spectrum of network activities such as TCP, UDP, and RTP/RTCP traffic. Tstat distinguishes between "complete" and "not complete" flows and also between clients (hosts that actively open a connection) and servers (hosts that passively listen for connection requests).

pegasus-kickstart. Compute jobs in Pegasus are wrapped using a lightweight C executable called pegasus-kickstart [36] that captures runtime job provenance data. The toolkit provides useful information about the execution of the wrapped task such

as (1) information about the execution node (architecture, OS, number of cores, available memory); (2) the environment setup of the machine while the task was running; (3) the task's characteristics and performance data (arguments, start time, duration, exit code, etc.); and (4) the task's output logs (`stdout` and `stderr`).

4. Online Monitoring System

In previous works [37, 38, 39], we have demonstrated the potential of workflow performance data capture and analysis to estimate workflow resource requirements and to detect anomalies present in a single workflow execution trace generated by the workflow management system. In this work, we aim to collect and correlate information from heterogeneous monitoring components that are resident in various execution sites. These monitoring sources include underlying data transfer infrastructure for workflows (i.e., Globus Online and Tstat) and I/O performance monitoring tools (i.e., Darshan). Each of these sources has its own proprietary format that introduces challenges related to how information can be integrated. By approaching the problem from a system architecture perspective (Figure 1), we use the RabbitMQ message broker [40] as the central endpoint where monitoring information is gathered from the various sources. Underneath the message broker, we have deployed a standard ELK stack. We use Logstash's RabbitMQ input plugin to fetch data from the AMQP endpoint and push it into an Elasticsearch instance. Additionally, we have built a custom workflow dashboard as a Kibana plugin. This dashboard allows users to select a particular workflow and visualize the related performance data in near real-time: both aggregated at a workflow level and a per job level.

In this section, we present an overview of the data collection framework. We describe the online monitoring capabilities provided by *pegasus-kickstart* [36], which are fundamental for workflow performance data gathering. We then describe the data collection challenges encountered and the subsequent solutions implemented to gather data from the monitoring components, followed by a description of the data capture flow during the execution of a workflow job. Finally, we present the Kibana plugin for data discovery and visualization.

4.1. Pegasus-Kickstart Online Monitoring

Near real-time monitoring enables the rapid detection of poor performance issues and workflow or infrastructure anomalies. By harvesting such information in a timely fashion, one could identify, mitigate, or prevent undesired behaviors at runtime. To this end, we have extended *pegasus-kickstart* to include fine-grained monitoring capabilities that can pull resource usage statistics of workflow running tasks within a predefined time interval. The maximum pulling frequency is limited to one second to prevent system flooding. This information is then published to an AMQP endpoint in JavaScript Object Notation (JSON) format so it can be ingested to a permanent storage (e.g., InfluxDB) or to an analysis framework (e.g., Elasticsearch). Table 1 summarizes performance metrics and workflow characteristics provided by *pegasus-kickstart*[1].

Field	Description
`event`	type of event (`kickstart.inv.online`)
`ts`	timestamp of the measurement
`hostname`	the hostname of the compute node
`site`	the execution site
`wf_uuid`	the workflow UUID
`wf_label`	the workflow label
`dag_job_id`	the job ID referring to Pegasus's dag
`xformation`	the job label from Pegasus's dag
`task_id`	task ID
`pid`	the process ID
`exe`	the invoked executable
`rank`	the process rank
`utime`	time spent on executing user code
`stime`	time spent on executing system code
`iowait`	time spent on waiting for IO
`vm`	virtual memory used by the process
`rss`	resident-set size memory used by the process
`procs`	number of processes
`threads`	number of threads
`bread`	number of bytes read
`bwrite`	number of bytes written
`rchar`	number of chars read
`wchar`	number of chars written
`syscr`	number of read system calls
`syscw`	number of write system calls
`bsend`	number of bytes sent
`brecv`	number of bytes received

Table 1: Summary of online workflow performance metrics and characteristics provided by *pegasus-kickstart*.

4.2. Data Collection

To support the online nature of our data capture system, we have designed and extended Pegasus to include online publishing capabilities. Specifically, we have modified the monitoring daemon (*pegasus-monitord*) and the transfer tool (*pegasus-transfer*). In addition to Pegasus-specific tools, we have also developed mechanisms to gather, preprocess, and publish job-level and infrastructure-level performance data from system profiling tools, such as Darshan and from transfer services such as Globus Online. Below is a summary of the challenges intrinsic to each tool and a description of our approach to overcome them.

pegasus-monitord. The Pegasus monitoring daemon follows a workflow execution and records provenance information from the workflow and its completed jobs. Typically, coarse-grained runtime provenance information is populated to a relational datastore (the Stampede Workflow Database [24]) upon job completion. To enable near real-time, fine-grained monitoring while still gathering and storing provenance data in a traditional

[1] Currently, this implementation is available online in a separate GitHub branch: https://github.com/pegasus-isi/pegasus/tree/panorama

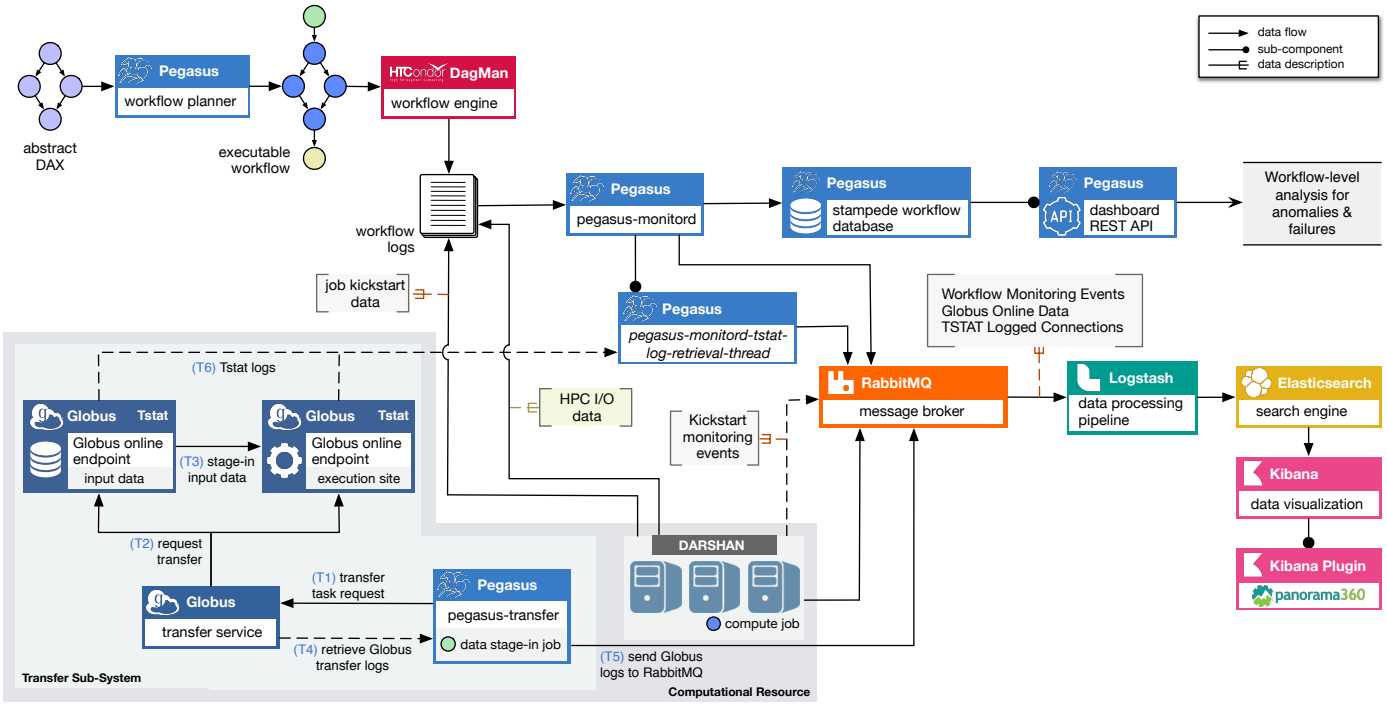


Figure 1: Architecture overview of the end-to-end online performance data capture and analysis process. How Panorama is using the Pegasus workflow management system to collect and publish workflow logs, execution traces and transfer logs.

relational database, we have developed a multiplexing capability that allows *pegasus-monitord* to publish events to an AMQP endpoint in addition to the relational database. We have also extended *pegasus-monitord* to parse additional monitoring events from job `stdout` records to facilitate population of events from Darshan.

pegasus-transfer. The Pegasus transfer tool performs data movement operations by invoking the appropriate underlying data transfer tool based on the protocol scheme specified for the source and destination URLs. It supports a variety of protocols such as SCP, GridFTP, HTTP, S3, stashcp, file copy, etc. [41, 42, 43]. Upon completion, transfer logs are summarized into file records (job's standard output files) which include the number of transfer operations performed, the amount of data transferred, and the transfer rate. Due to the imminent need for a better and more rapid detection of data transfer issues, we have extended *pegasus-tranfer* to perform transfer operations via the Globus Online transfer service. Not only does it provide fast, secure, and reliable data movement operations for workflow data transfers, but Globus Online also provides a querying mechanism to retrieve the transfer status near real-time. After a transfer finishes, either with a successful or a failed status, *pegasus-transfer* queries the Globus transfer service and publishes detailed information about the transfer to the AMQP endpoint. Fine-grained transfer logs include:

- Transfer request, start, and completion times;
- Transfer steps (for long running transfers);
- Accurate transfer throughput;
- Level of concurrency and parallelism used in transfer;
- Number of subtasks failed and retried; and
- Human readable error messages describing the failure.

Darshan. We use the Darshan profiling tool to gain insights into the I/O performance of Message Passing Interface (MPI) applications. Darshan is usually enabled by default on large HPC systems and its logs are available in a proprietary binary format at a standard location for each job when it completes. We have developed a new tool called *pegasus-darshan* that: (1) determines the corresponding log file for a particular job based on the local resource manager's job ID; and (2) parses the binary file and generates a JSON record containing the relevant Darshan statistics. This tool is invoked at the end of each MPI remote job execution, and the extracted Darshan information gets encoded in the job's `stdout`. The JSON record containing information from Darshan (Listing 1) is parsed by *pegasus-monitord* and is published to the AMQP endpoint. Currently, we only extract a subset of the available Darshan statistics (summarized in Table 2).

Tstat. Execution sites often do not provide direct access to Tstat logs. Instead, execution sites make a subset of preprocessed records available to their users. For instance, at the National Energy Research Scientific Computing Center (NERSC), Tstat processed logs are published to an ELK cluster, which users can access via a Kibana dashboard. One thing to note is that not all Tstat records are made available in the ELK cluster (NERSC filters records where transfer throughput is below 100 MB/s, i.e. transfers from/to slow speed networks are omitted). Due to the delay (up to several hours) in preprocessing the large amount of transfer logs, online monitoring becomes challenging. Finally, Tstat captures very low-level network statistics that do not contain metadata associating TCP flows to a particular data transfer. Such disjointed information from TCP flow fine-grained statistics and data transfers hinders the ability to identify issues

Field	Description
event	event type: stampede.task.monitoring
monitoring_event	monitoring event type: darshan.perf
darshan_log_version	log file version number
exe	name of the executable
uid	user id that job ran as
jobid	job id from the scheduler
start_time	start time of the job
end_time	end time of the job
nprocs	number of MPI processes
run_time	run time of the job in seconds
STDIO.*	STDIO module data
POSIX.*	POSIX module data

Table 2: Summary of Darshan metrics captured during workflow execution. Since Darshan logs are only produced at job completion, near real-time monitoring is attained at per job completion granularity.

concerning a specific transfer operation. The problem becomes more complex in large-scale production environments, where hundreds of transfers may occur simultaneously.

Data collection with Tstat is currently performed manually at the execution site, but efforts are already in place to enable the automation of the retrieval process. Globus uses GridFTP for file transfers and spawns concurrent GridFTP processes to transfer multiple files simultaneously. Furthermore, each GridFTP process uses multiple TCP connections [41, 44], and connections are reused for all files within a process. In order to correlate individual file transfers with TCP flows, we have to leverage Globus features to match specific transfer tasks to Tstat statistics. To achieve this, we are currently extending the Globus Transfer service to expose low level network information (e.g., GridFTP server IPs, and TCP flows ports) regarding data transfers including individual file transfers.

4.3. A Versatile Approach for Integrating New Tools

As monitoring tools are constantly evolving, we argue it is crucial to provide a way to seamlessly integrate new tools into the overall monitoring and analysis system. To this end, we have extended *pegasus-monitord* with the ability to parse additional monitoring events from job `stdout` records. This feature allows us to create job wrapper scripts that can invoke arbitrary monitoring tools and append statistics produced by these tools to the `stdout` upon task completion. These statistics have to be in JSON format and must be wrapped within an output segment specified by keyword tags that indicate the start and end of a record. While *pegasus-monitord* is processing `stdout` records, it is able to identify this special segment and trigger a monitoring event after parsing the JSON document. By following this approach, the monitoring data will be added to Elasticsearch under the index hosting the workflow events by default. However, by using Logstash, these events can be filtered, preprocessed, and finally ingested into a new custom index. In our architecture, this approach is being used to add the data produced by Darshan on the remote execution site to Elasticsearch as described in Section 4.2. A generic monitoring payload consists of a monitoring event, a payload, and a timestamp. An example of a generic monitoring payload can be found in Listing 1.

```
@@@PEGASUS_MONITORING_PAYLOAD - START @@@
{
  "monitoring_event": "darshan.perf",
  "payload": [
    {
      "POSIX_module_data": {...},
      "STDIO_module_data": {...},
      "compression_method": "ZLIB",
      "darshan_log_version": "3.10",
      "end_time": 1531941742,
      "end_time_asci": "Wed Jul 18 19:22:22 2018",
      "exe": "namd2 equilibrate.conf",
      "jobid": "1547",
      "metadata": {
        "h": "romio_no_indep_rw=true;cb_nodes=4",
        "lib_ver": "3.1.6"
      },
      "nprocs": 8,
      "run_time": 65.0,
      "start_time": 1531941678,
      "start_time_asci": "Wed Jul 18 19:21:18 2018",
      "uid": "1003"
    }
  ],
  "ts": 1531941740
}
@@@PEGASUS_MONITORING_PAYLOAD - START @@@
```

Listing 1: Generic monitoring payload example.

4.4. Data Capture Flow

To convey application and infrastructure performance data from various heterogeneous data sources to the workflow's end user in a comprehensive and coherent fashion, we have designed and implemented the flow illustrated in Figure 1. The flow entry point is a Pegasus workflow, and performance information collected during the workflow execution is made accessible through a custom Kibana dashboard or the Pegasus dashboard REST API.

On a submit host, Pegasus takes in a high-level description of the user workflow and generates an executable workflow that is managed by HTCondor DAGMan. DAGMan releases jobs when they are ready for execution to the local HTCondor scheduler that in turn submits the jobs to remote resources for execution. On a remote resource, the jobs are launched by *pegasus-kickstart*, which monitors job execution and sends online monitoring events to the RabbitMQ message broker. When a compute job is an MPI job, the job wrapper automatically invokes *pegasus-darshan* to parse I/O characteristics from Darshan logs upon completion of the job and encodes them as part of the job's standard output (`stdout`). The job `stdout` is then included in the *pegasus-kickstart* output, which is transferred back automatically to the workflow submit host. On the submit host, the job information and the DAGMan logs are parsed by *pegasus-monitord*, which publishes this information to the RabbitMQ message broker and populates the Pegasus Stampede workflow database. Workflow data transfer jobs are executed using *pegasus-transfer*. While managing the transfers, *pegasus-transfer* initiates transfer requests to the Globus transfer service and frequently (at a predefined time interval) performs pull requests inquiring about the status of the transfers.

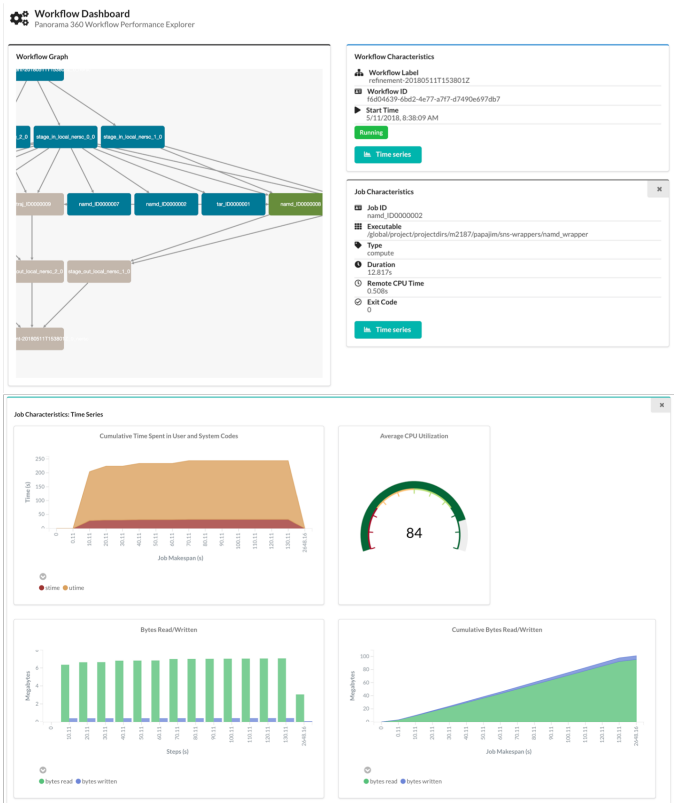


Figure 2: Screenshots of the Kibana plugin for near real-time monitoring of workflow performance metrics. *Top:* workflow progression and detailed job characteristics. *Bottom:* time series data of job performance.

Upon completion, Globus transfer logs are retrieved and published by *pegasus-transfer* to the RabbitMQ message broker. All the online monitoring data published to RabbitMQ are automatically fed into Elasticsearch using the Logstash connector, and afterwards are accessible in the custom Kibana dashboard plugin. Additionally, workflow and job level data recorded in the relational workflow database (Pegasus Stampede [24]) is made available via the Pegasus dashboard REST API.

4.5. Data Discovery and Visualization

The goal of this work is to provide a resource for the collection, discovery, analysis, and sharing of end-to-end performance data of scientific workflow executions. In our proposed system, data archiving and browsing are managed as follows. Structured data is stored in a traditional relational database (Pegasus Stampede) that permanently records workflow events and statistics. This data can then be accessed via a REST API to query the workflow status and the job-specific or workflow-specific performance metrics during execution. Online performance data, as well as workflow events, are also stored in an unstructured format in Elasticsearch. By generating inverted indexes for every field in the data (inverted indexes can be used simultaneously in queries), data discovery is empowered by full text search, which allows for the unveiling of hidden knowledge (e.g., differentiate identical error codes using the ascii error messages). For our experiments, the Elasticsearch deployment is centralized in a single server. As data ingestion and querying traffic become larger, we plan to enable the distributed deployment (shards) of Elasticsearch.

To enable near real-time monitoring and easy, interactive visualization of workflow characteristics and performance metrics, we have developed an open-source Kibana plugin [45] (Figure 2). This dashboard combines and summarizes data gathered from the various sources described in this paper and displays concise information (in the form of tables and plots) about the workflow execution at runtime. The goal of this dashboard is to allow users to effortlessly pinpoint performance issues without needing to dig into the hundreds of MBs of execution logs generated during execution. As events occur and their data are pushed into Elasticsearch, the user can follow the progression of the workflow in near real-time. Statistical or Machine Learning (ML) analyses on the data can be performed by simply running one or a few of the multitude of Kibana plugins freely available online, all within the same platform. Our plugin is a live product and is constantly evolving with new capabilities. The current collection of available plots includes time-series analysis of workflow-level and job-level statistics such as CPU utilization, I/O read and write operations, I/O wait, I/O throughput and runtime, among others (Figure 2).

```
pegasus.monitord.encoding = json

pegasus.catalog.workflow.amqp.url = \
amqp://[username:password]@hostname[:port]/exchange_name

pegasus.catalog.workflow.amqp.events = stampede.*
```

Listing 2: Enabling Pegasus Stampede events via the Pegasus' properties file.

5. Deploying the Online Monitoring Architecture

In this section, we describe how the proposed architecture can be deployed. Our architecture allows users to partially or fully enable monitoring features by simply providing specific Pegasus profiles or properties as described below. The source code is publicly available on GitHub [46].

Enabling Stampede Events. In order to direct *pegasus-monitord* to publish all of its events to the AMQP endpoint in JSON format, three properties must be specified in the workflow's properties file (e.g., `pegasus.properties`): (1) `pegasus.monitord.encoding` enables the JSON output format; (2) `pegasus.catalog.workflow.amqp.url` contains the connection information to the AMQP endpoint; and (3) `pegasus.catalog.workflow.amqp.events` filters which events should be published. Listing 2 shows an example of these properties in practice.

Enabling Transfer Events. To enable the mechanisms to publish transfer statistics from the Globus Transfer Service to an AMQP endpoint in JSON format, two Pegasus profiles must be specified in the workflow's sites catalog (e.g., `sites.xml`), under the site where *pegasus-transfer* will be invoked (e.g., `local`), as shown in Listing 3. The environment variable `PEGASUS_TRANSFER_PUBLISH` operates as an on/off switch to the

transfer monitoring, and the PEGASUS_AMQP_URL variable provides the AMQP endpoint definition to *pegasus-transfer*.

```
<site handle="local">
  ...
  <profile namespace="env" key="PEGASUS_TRANSFER_PUBLISH">
    1
  </profile>
  <profile namespace="env" key="PEGASUS_AMQP_URL">
    amqp://[username:password]@hostname[:port]/exchange_name
  </profile>
</site>
```

Listing 3: Enabling Pegasus Transfer events via the Pegasus' sites catalog.

Enabling Kickstart Online Traces. To publish traces of resource usage statistics, two Pegasus profiles must be specified in the workflow's sites catalog (e.g., `sites.xml`) under the compute site (Listing 4): (1) `pegasus.gridstart.arguments` instructs *pegasus-kickstart* to collect resource usage statistics every N seconds; while (2) KICKSTART_MON_URL points to AMQP's rest api for publishing data, which looks similar to "`api/exchanges/exchange_name/publish`". This way, *pegasus-kickstart* will push such information to an AMQP endpoint in JSON format.

```
<site handle="compute">
  ...
  <profile namespace="pegasus" key="gridstart.arguments">
    -m interval_seconds
  </profile>
  <profile namespace="env" key="KICKSTART_MON_URL">
    rabbitmq://[USERNAME:PASSWORD]@hostname[:port]/...
  </profile>
</site>
```

Listing 4: Enabling Kickstart online traces via the Pegasus' sites catalog.

Enabling Darshan Statistics. As mentioned in Section 4, we use a wrapper script to retrieve Darshan logs. This script identifies the location of the generated Darshan logs and invokes *pegasus-darshan* after the completion of the MPI job. The "pegasus-darshan" tool parses the Darshan logs and outputs a monitoring payload (Listing 1). The propagation of these events depends on whether the Stampede events have been enabled. An example of the wrapper script, which was used to retrieve Darhan logs from Cori at NERSC, is shown in Listing 5.

Setting Up The Monitoring Backend. Configuring the monitoring backend (RabbitMQ, Elasitcsearch, Logstash, and Kibana) can turn out to be a cumbersome and challenging process, especially if one is not familiar with these tools and only wants to collect and analyze workflow execution statistics. By leveraging container technologies (Docker [47] and Docker Compose [48]), we have developed a container orchestration mechanism that spins up all the required services preconfigured to capture all events produced by the workflow execution. Additionally, this automation spins up a version of Kibana that has the Panorama plugin installed. All container services use persistent volumes, which consequently maintain the state of the daemons and the collected data between restarts. Finally, all services can be triggered using a single command (`docker-compose up -d`), which is reliable and easy to use [49].

```
#!/bin/bash -l

srun $PEGASUS_HOME/bin/pegasus-monitor namd2 "$@"

#post job parse darshan output
DAY=$(date '+%d')
DAY=${DAY##0}
MONTH=$(date '+%m')
MONTH=${MONTH##0}
YEAR=$(date '+%Y')

darshan_base=${DARSHAN_LOGDIR}/${YEAR}/${MONTH}/${DAY}
darshan_file=\
${darshan_base}/${SLURM_JOB_USER}_*${SLURM_JOB_ID}_*.darshan

for f in $darshan_file; do
   $PEGASUS_HOME/bin/pegasus-darshan -f "$f"
done
```

Listing 5: Example of a wrapper script for gathering Darshan statistics from Cori at NERSC.

6. Case Studies

In this section, we present case studies with two real world scientific workflow applications: a CPU-intensive material science application and a data-intensive genomics application. The goal of these case studies is to demonstrate the ability of our system to accurately capture performance metrics that are critical for improving the efficiency and resilience of current and upcoming systems and scientific workflow applications. We experiment within a controlled environment, where anomalies are injected at runtime, so that we can measure whether our system adequately captures such behaviors.

6.1. Scientific Applications

Spallation Neutron Source. We use a material, science-related workflow developed at the Spallation Neutron Source (SNS) [50], a DOE research facility at Oak Ridge National Laboratory. The SNS workflow executes an ensemble of molecular dynamics (MD) and neutron scattering intensity calculations to optimize a model parameter value, for example, to investigate temperature and hydrogen charge parameters for models of water molecules. The workflow takes as input a set of temperature values and four additional parameters: (1) type of material, (2) the number of required CPU cores, (3) the number of timesteps in the simulation, and (4) the frequency at which the output data is written. Figure 3 shows a branch of the workflow that analyzes a single temperature value. First, each set of parameters is fed into a series of parallel molecular dynamics simulations using NAMD [51]. The first simulation computes an equilibrium, which is used by the second simulation to compute the production dynamics. The output from the MD simulations has the global translation and rotation removed using AMBER's [52] *cpptraj* utility, which is passed into Sassena [53]

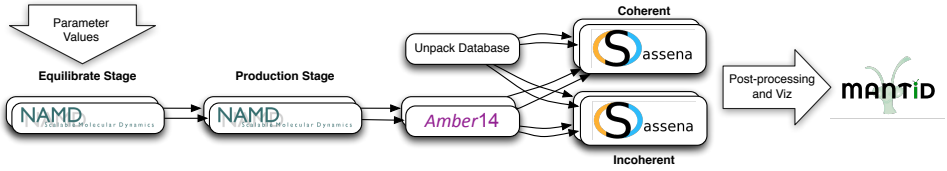


Figure 3: A diagram of a branch of the SNS workflow.

to compute coherent and incoherent neutron scattering intensities from the trajectories. The final outputs of the workflow are transferred to the user's desktop and loaded into Mantid [54] for analysis and visualization. In our experiments, we configured the SNS workflow to spawn 8 MPI processes for the Equilibrate stage, 16 MPI processes for the Production stage, and 16 MPI processes for both Coherent and Incoherent neutron scattering intensities calculations. Unpacking the Sassena DB and executing *cpptraj* were done using a single core only.

1000Genome. The 1000 genomes project provides a reference for human variation, having reconstructed the genomes of 2,504 individuals across 26 different populations [55]. The test case used in this work identifies mutational overlaps using data from the 1000 genomes project in order to provide a null distribution for rigorous statistical evaluation of potential disease-related mutations [56]. This test case (Figure 4) is composed of five different tasks: (1) *individuals* – fetches and parses the Phase 3 data from the 1000 genomes project per chromosome; (2) *populations* – fetches and parses five super populations (African, Mixed American, East Asian, European, and South Asian) and a set of all individuals; (3) *sifting* – computes the SIFT scores of all of the SNPs (single nucleotide polymorphisms) variants, as computed by the Variant Effect Predictor; (4) *pair overlap mutations* – measures the overlap in mutations (SNPs) among pairs of individuals; and (5) *frequency overlap mutations* – calculates the frequency of overlapping mutations across subsamples of certain individuals. In order to fit an instance of the workflow execution into our testbed (see description below), we are processing 2 chromosomes for which we have pruned the original datasets to about 10% (about 1GB each) of the original data (about 11GB per individual dataset). For this experiment, each workflow is composed of 22 individuals jobs, 2 sifting jobs, 14 frequency overlap mutations jobs, and 14 pair overlap mutations jobs.

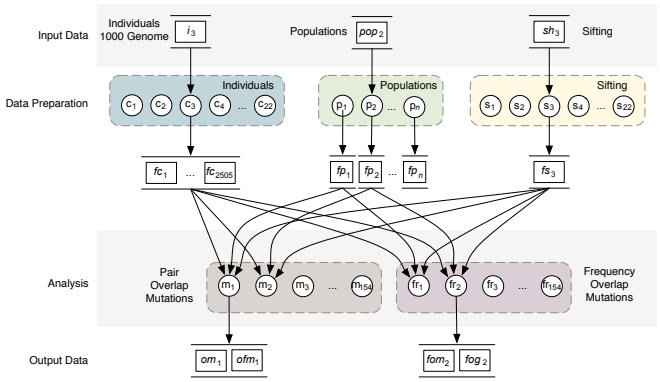


Figure 4: Overview of the 1000Genome sequencing analysis workflow.

6.2. Experimental Setup

Figure 5 presents the experimental system on the ExoGENI cloud testbed [57]. It is orchestrated over a federation of independent cloud sites located across the US and connected via national research circuit providers such as Internet2 [58] and ESNet [59], through their programmable exchange points. ExoGENI provides users with isolated virtual compute, storage, and network resources named "slices" of infrastructure. ExoGENI uses its native ORCA (Open Resource Control Architecture) [60] control framework software to offer a unified hosting platform for deeply programmable, multi-domain cloud applications. By using a controlled execution environment such as ExoGENI, we ensure that system interference are minimized as synthetically generated interference affect specific performance metrics targeted for evaluation.

Our setup consisted of one data node, one master node, and four compute (worker) nodes. Each node had 4 vCPUs clocked at 2.2 Ghz, 10GBytes of RAM, and 75GBytes of storage. Both the compute and the master nodes were collocated on the same rack, while the data node was spawned on a rack in another region. The master and the compute nodes communicated via the rack switch, while the data node could be reached via ESnet's network. To facilitate the execution of the workflows, we configured our slice with HTCondor and Slurm, and we configured Pegasus on the master/submit node (where HTCondor managers and schedulers, as well as Slurm master, reside). Globus Endpoints were created on both the master and the data nodes. Additionally, the master node and the compute nodes had access to a shared file system (NFS), which was physically located on master's hard disk.

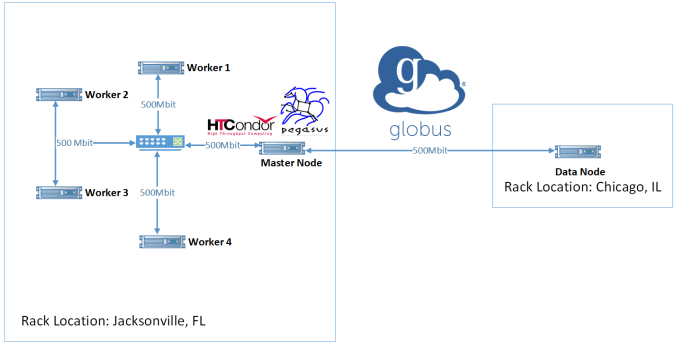


Figure 5: Experimental setup on the ExoGENI testbed.

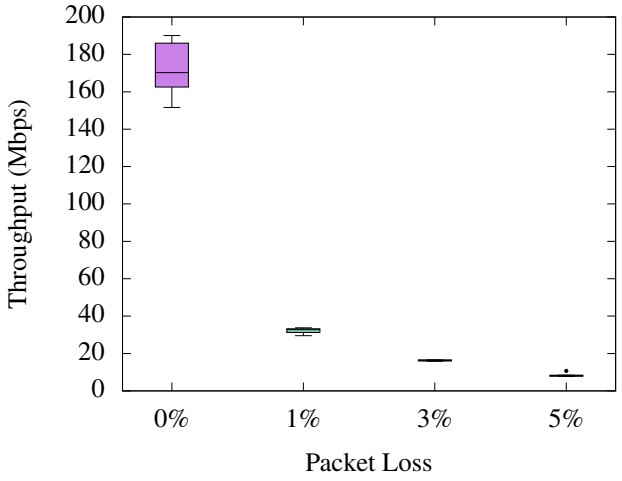


Figure 6: Transfer throughput of 1000Genome workflow with packet loss.

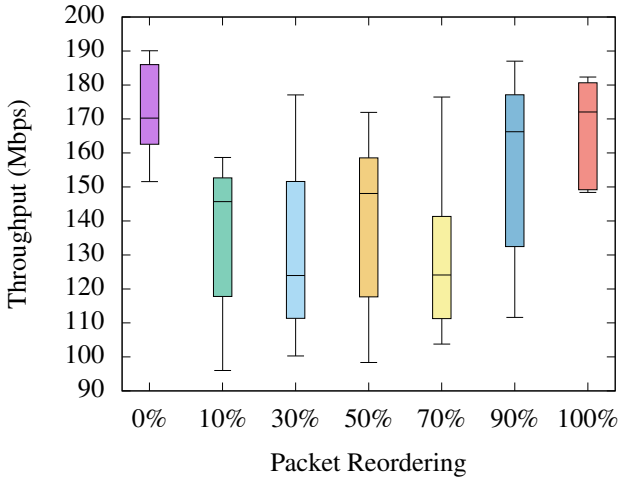


Figure 7: Transfer throughput of 1000Genome workflow with packet reordering.

6.3. Network Throughput

In this set of experiments, we aim to demonstrate the ability of our framework to capture network anomalies due to low performance in a network. More specifically, we arbitrarily inject synthetic packet loss and packet reordering anomalies during the execution of workflow transfer jobs so that we can assess whether the information captured by our framework at runtime is sufficient to show discrepancies due to network anomalies.

For this experiment, we performed runs of the 1000Genome workflow where the input data is staged in from the data node to the master node (the execution site) with Globus transfer. Job scheduling, execution, and data transport between the master and compute nodes are performed through HTCondor. Upon job completion, output data is staged out to the data node using Globus transfer service. For each scenario described below, we performed six runs of the 1000Genome workflow to insure statistical significance (error below 5%).

We used the Linux native Traffic Control (TC) [61] toolset to configure the Linux kernel packet scheduler so that we could introduce synthetic network and I/O anomalies, including delay, packet loss, and jitter, among others. Figure 6 shows the effect of packet loss when using TC to introduce a randomized percentage of packet losses at rates of 1%, 3%, and 5%. This interference is generated during the execution of a Pegasus stage-in transfer job (`stage_in_0_1`) for the 1000Genome workflow, which transfers 790 MB of input data via ESNet (from the data to the master node). When there is no interference in the network connection, the mean transfer throughput is about 170 Mbps. In the event of randomized packet loss disturbances, network throughput significantly degrades by up to ~6x – measured throughput is about 32, 18, and 9 Mbps for 1%, 3%, and 5% packet loss, respectively.

Although TCP attempts to mitigate out-of-order delivery of data packets, this is still a common issue in today's computational systems [62]. Not only does this impose significant computational overheads on hosts, but it also impacts the throughput of TCP. Therefore, having the ability to accurately identify and assess such impact on workflow job executions is crucial to support the decision process of performing actions to prevent or mitigate such effect. To this end, we inject synthetic packet reordering by arbitrarily delaying some of the packets for 10 ms before dispatching them. For example, 30% reordering means 30% of packets are delayed for 10 ms, while the remaining packets are dispatched immediately with a correlation of 50%.

Figure 7 depicts distribution measurements of the network throughput for 10%, 30%, 50%, 70%, 90%, and 100% packet reordering. Transfer throughput has a noticeable slowdown when the packet reordering rate increases from 0% to 30% (~0.2x degradation). Intriguingly, the network throughput raises for reordering at a 50% rate – as more packets are delayed, the more ordered they become. Similar behavior can also be observed in the throughput for reordering rates of 70% and 90%, which are symmetrical to the throughput of 30% and 10%, respectively.

Both of the network anomalies above have significant impact on the transfer throughput. However, it is difficult to identify the issue that degrades network throughput from the Globus transfer logs alone. Therefore, Tstat becomes a centerpiece of our data capture architecture as it provides fine-grained low-level network statistics of TCP flows. By combining both Globus transfer and Tstat logs, we can derive a more complete picture of the data transfers, which is fundamental for end-to-end monitoring of workflow executions.

6.4. I/O Throughput

A typical bottleneck when running large-scale, data-intensive workflows is the heavy use of disk I/O. As parallel file system performance is not keeping up with compute and memory performance, it is imperative to properly identify situations where low I/O performance may severely impact the workflow makespan. To attenuate this issue, both *in situ* and *in transit* solutions have been used to accelerate the workflow performance [63, 64].

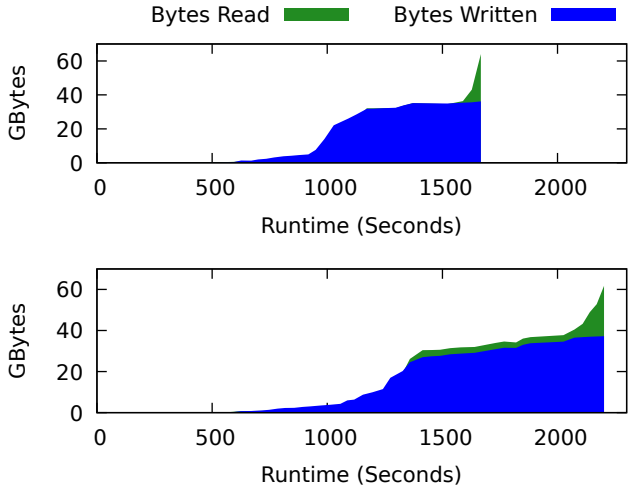


Figure 8: Cumulative I/O over time. *Top:* 1000Genome workflow without interference. *Bottom:* I/O stressing of the workers.

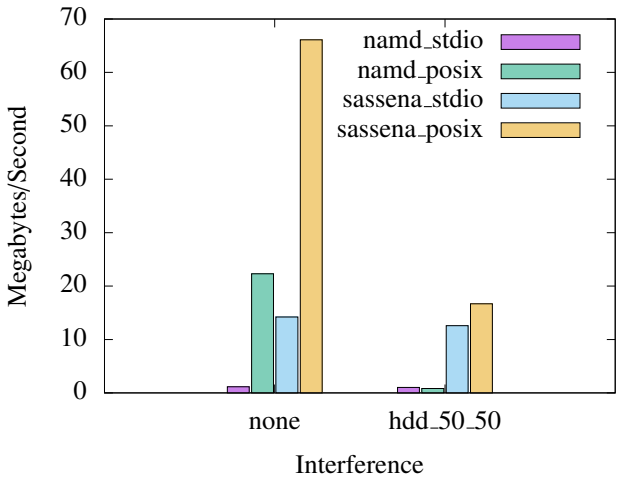


Figure 9: Average STDIO and POSIX performance for NAMD and Sassena Jobs obtained from Darshan's logs.

To evaluate the ability of our framework to capture fine-grained I/O information, we used Stress [65], a simple workload generator that can impose configurable amount of CPU, memory, I/O, and disk stress on the system. Figure 8 shows the cumulative reads and writes for the 1000Genome workflow over time, with and without disk stress (*top*: regular workflow execution with no interference; *bottom*: workflow execution with disk stress). To introduce the interference, we spawned one stress process on each worker (compute) node that performed about 50 MB of I/O writing to the node's disk continuously. Since workflow tasks have dependencies, i.e., one child job does not start its execution until all its parents have completed, the workflow is slowed down by a factor of ~1.5 – i.e., disk stress degrades the node's disk throughput, thus jobs require more time to complete I/O operations.

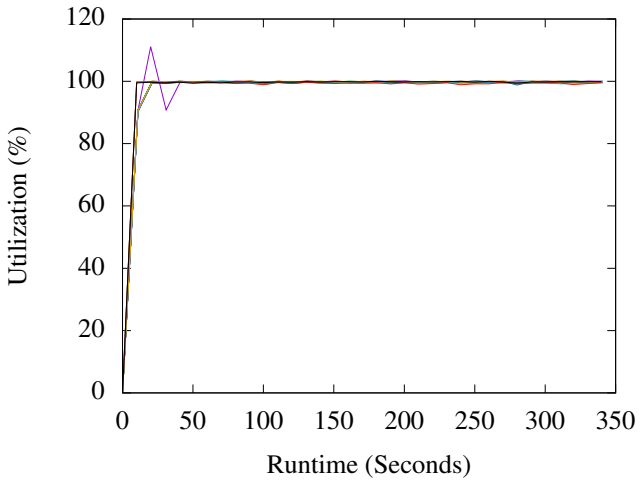
The above performance metric is obtained with *pegasus-kickstart*, which provides time series data of I/O read and write operations at runtime. Although this metric aids in pinpointing bottlenecks in the workflow, it lacks fine-grained information regarding jobs' I/O profiles. As previously mentioned, Darshan provides I/O characterizations for HPC applications including properties such as patterns of access within files. By combining Darshan and Pegasus logs, one can accurately identify the bottlenecks or low performance issues due to I/O operations at different levels (e.g., single or parallel operations).

For this experiment, we used the SNS workflow, which has four parallel (MPI) jobs in its pipeline and is instrumented with Darshan[2]. Similarly to the 1000Genome workflow execution, we stage in the workflow input data from the data node to the master node via Globus transfer, which is then stored in a parallel file system (NFS) accessible to the worker nodes. Computing jobs (NAMD and Sassena) are submitted to HTCondor (used as a broker), which dispatches them to Slurm, so that MPI jobs can benefit from all available resources. Note that in the parallel execution, multiple processes may write to a single file simultaneously. In the evaluated scenario, all processes write to a single file in the parallel file system, thus multiple I/O write/read requests may happen at the same time. Therefore, on the master node we spawned two stress processes that write ~50 MB each, and there was no disk stress on the worker nodes. Figure 9 shows the parallel file system performance, captured by Darshan, during the execution of the SNS' NAMD and Sassena jobs. The performance of STDIO operations is not affected by the interference. On the other hand, POSIX performance is severely impacted – I/O throughput degradation is slowed down up to a factor of 3 when compared to a regular execution with no interference.

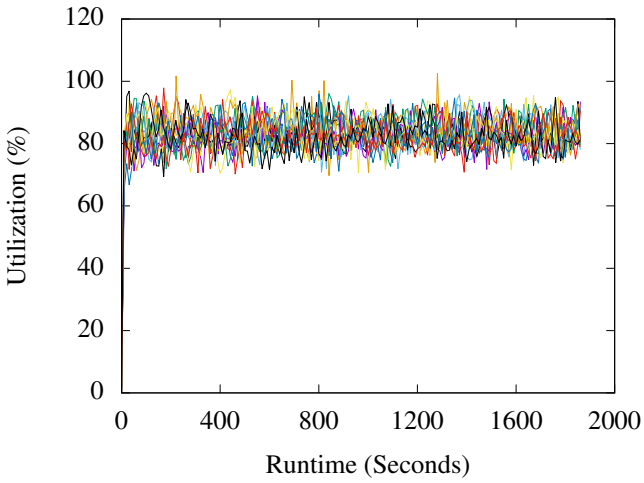
6.5. CPU Contention

External load is a common factor that often negatively impacts the workflow makespan. In shared environments such as grids, external processes (including processes spawned by different users sharing the same resource) may substantially impact CPU performance. We have performed CPU stress tests on one CPU per worker node during the execution of the SNS workflow. Figures 10a and 10b show the CPU utilization per rank of the NAMD MPI job without and with interference, respectively. Under no interference, CPU utilization for each MPI rank is nearly 100%, while in the scenario with interference CPU utilization resonates between 70% and 90%. This is due to the stress processes not having been pinned to a specific core of each node with affinity, while each MPI process had affinity set during submission. Notice that due to CPU utilization degradation, the NAMD job resulted in a slowdown by a factor of 5 – most probably because NAMD largely depends on the performance of work synchronization between tasks of the MPI jobs.

[2]For additional information refer to Darshan's documentation https://www.mcs.anl.gov/research/projects/darshan/documentation/



(a) without interference.



(b) with interference.

Figure 10: CPU utilization per rank for the NAMD MPI job. Without interference the CPU utilization is steady, close to 100%. However with interference CPU utilization fluctuates between 70% and 90%. To introduce interference, one stress process per worker consumed approx. 25% of the node's CPU time.

7. Open Access Data

Service	Address
Elasticsearch	https://data.panorama.isi.edu
Kibana	https://kibana.panorama.isi.edu

Table 3: Open access data services.

To enable collaborative and reproducible research, and in an attempt to provide a comprehensive set of end-to-end workflow characteristics and performance behaviors, we have taken steps towards the development of an open access repository for end-to-end workflow statistics. Currently, the repository hosts workflow data collected in the scope of this work, but we aim to enrich it with additional workflow runs from a variety of scientific domains in the near future. The current deployment exposes an Elasticsearch and a Kibana instance that can be accessed via the URLs shown in Table 3. These services are publicly accessible, but read-only. Kibana accesses the JSON documents stored in the Elasticsearch instance, and by using a web-browser and the Panorama 360 plugin, one can explore the available data in a user-friendly way. On the other hand, Elasticsearch is accessible only programmatically via the REST API. Table 4 shows the available indexes used to organize the JSON documents. Due to the restrictions we have applied to enforce read-only access, API requests targeting indexes that are not defined in Table 4 will be rejected with an access denied error message. A common approach to retrieve and process data locally from the Elasticsearch endpoint is by using Python scripts. An example of a simple script is shown in Listing 6.

Index	Description
`panorama_transfer`	Logs retrieved from the Globus transfer service
`panorama_kickstart`	Resource utilization traces collected by pegasus kickstart online
`panorama_stampede`	Workflow execution events and generic monitoring events, such as Darshan's performance events

Table 4: Open access Elasticsearch indexes.

```python
#!/usr/bin/env python

from elasticsearch import Elasticsearch

workflow_id = "3f1101db-b220-4155-84e3-ea85624ee82f"

es = Elasticsearch("https://data.panorama.isi.edu")

query = "xwf__id: \"" + workflow_id + "\""
res = es.search(index="panorama_stampede",q=query,size=100)
print res['hits']['total']

query = "wf_uuid: \"" + workflow_id + "\""
res = es.search(index="panorama_transfer",q=query,size=100)
print res['hits']['total']

query = "wf_uuid: \"" + workflow_id + "\""
res = es.search(index="panorama_kickstart",q=query,size=100)
print res['hits']['total']
```

Listing 6: Querying open access data.

8. Conclusions and Future Work

In this work, we have presented a framework to orchestrate a number of well-established and newly developed state-of-the-art tools in order to capture and correlate fine-grained and coarse-grained information about scientific workflow executions from various, heterogeneous sources, in an online manner. Such sources include network (Globus, Tstat), filesystem (Darshan, kickstart), and compute resources (kickstart). To this end, we have extended various components (*pegasus-monitord*, *pegasus-transfer*) of Pegasus WMS to enable online performance monitoring. Moreover, we have developed new tools

(*pegasus-darshan*) to facilitate performance data collection on remote nodes. Additionally, we presented a custom Kibana plugin, tailored to the needs of tracking a workflow execution and identifying potential issues at runtime. We evaluated our approach by executing normal and anomalous runs of two different classes of workflows in a controlled environment. Our experiments demonstrate that this architecture is able to accurately collect relevant performance metrics that can then be used to identify and analyze performance issues. The data collected is a useful resource of training data for automated machine learning analysis.

In the future, we plan to create a way to methodically collect Tstat data from the execution sites (that correspond to a workflow's data transfers) and to continue improving the custom Kibana plugin by adding new functionality and visualizing more aspects of a workflow's execution. Additionally, we plan to extend the types of resources that we collect data about. Currently, our framework does not retrieve statistics from accelerators, such as GPUs and FPGAs. Since, heterogeneous hardware is becoming more and more common in computing centers, we plan to explore efficient ways to include metrics produced by these hardware components into our architecture. Finally, we plan to provide a mechanism than can track all the data's original sources in order to enable offline analysis and data ingestion into other frameworks. We aim to introduce a new database that will be automatically populated by Pegasus while harvesting logs for performance statistics. This database will be storing metadata of the collected performance statistics, describing their original source location and the workflow that are associated with, so users can retrieve workflow performance data even when the live data collection fails or Elasticsearch gets corrupted. This work is still in early stages.

Acknowledgments

This work was funded by DOE contract #DESC0012636, "Panorama—Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows", and U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357. We thank G. Juve and D. Król for their contributions on extending pegasus-kickstart and enabling online monitoring. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract #DE-AC02-05CH11231.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, Workflows for e-Science: Scientific Workflows for Grids, Springer Publishing Company, Incorporated, 2014.

[2] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger, Pegasus: a workflow management system for science automation, Future Generation Computer Systems 46 (2015) 17–35. doi:10.1016/j.future.2014.10.008.

[3] U. D. o. E. Advanced Scientific Computing Research, Analytical Modeling Projects, https://science.energy.gov/ascr/research/next-generation-networking/am-projects/ (2018 (accessed August 3, 2018)).

[4] E. Deelman, C. Carothers, A. Mandal, B. Tierney, J. S. Vetter, I. Baldin, C. Castillo, G. Juve, D. Krol, V. Lynch, B. Mayer, J. Meredith, T. Proffen, P. Ruth, R. Ferreira da Silva, PANORAMA: An approach to performance modeling and diagnosis of extreme scale workflows, International Journal of High Performance Computing Applications 31 (1) (2017) 4–18. doi:10.1177/1094342015594515.

[5] Z. Liu, I. Foster, The Robust Analytic Models for Science at Extreme Scales project, https://ramsesproject.github.io (2018 (accessed August 3, 2018)).

[6] ELK stack, https://www.elastic.co/elk-stack (2018).

[7] M. C. Calzarossa, L. Massari, D. Tessera, Workload characterization: A survey revisited, ACM Computing Surveys (CSUR) 48 (3) (2016) 48.

[8] D. G. Feitelson, D. Tsafrir, D. Krakov, Experience with using the parallel workloads archive, Journal of Parallel and Distributed Computing 74 (10) (2014) 2967–2982.

[9] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. H. Epema, The grid workloads archive, Future Generation Computer Systems 24 (7) (2008) 672–686.

[10] D. Kondo, B. Javadi, A. Iosup, D. Epema, The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 398–407.

[11] D. Krol, R. Ferreira da Silva, E. Deelman, V. E. Lynch, Workflow performance profiles: Development and analysis, in: Euro-Par 2016: Parallel Processing Workshops, 2016, pp. 108–120. doi:10.1007/978-3-319-58943-5_9.

[12] M. L. Mondelli, M. T. de Souza, K. Ocaña, A. de Vasconcelos, L. M. Gadelha Jr, Hpsw-prof: a provenance-based framework for profiling high performance scientific workflows, in: Proceedings of Satellite Events of the 31st Brazilian Symposium on Databases (SBBD 2016), SBC, 2016, pp. 117–122.

[13] R. Ferreira da Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling and evaluating research on scientific workflows, in: 10th IEEE International Conference on e-Science, eScience'14, 2014, pp. 177–184. doi:10.1109/eScience.2014.44.

[14] D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, C. Goble, Common motifs in scientific workflows: An empirical analysis, Future Generation Computer Systems 36 (2014) 338–351.

[15] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Generation Computer Systems 29 (3) (2013) 682–692.
URL http://pegasus.isi.edu/publications/2013/JuveG-Characterizing.pdf

[16] R. Ferreira da Silva, T. Glatard, A science-gateway workload archive to study pilot jobs, user activity, bag of tasks, task sub-steps, and workflow executions, in: I. Caragiannis, et al. (Eds.), Euro-Par 2012: Parallel Processing Workshops, Vol. 7640 of Lecture Notes in Computer Science, 2013, pp. 79–88. doi:10.1007/978-3-642-36949-0_10.

[17] L. Ramakrishnan, D. Gannon, A survey of distributed workflow characteristics and resource requirements, Indiana University (2008) 1–23.

[18] S. Ostermann, R. Prodan, T. Fahringer, A. Iosup, D. Epema, A trace-based investigation of the characteristics of grid workflows, in: From Grids to Service and Pervasive Computing, Springer, 2008, pp. 191–203.

[19] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, A. Iosup, The workflow trace archive: Open-access data from public and private computing infrastructures, IEEE Transactions on Parallel and Distributed Systems (2020) 1–1Funding Acknowledgments: NSF 1664162. doi:10.1109/TPDS.2020.2984821.

[20] National Energy Research Scientific Computing Center (NERSC), https://www.nersc.gov.

[21] Oak Ridge Leadership Computing Facility (OLCF), https://www.olcf.ornl.gov.

[22] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, N. Wilkins-Diehr, Xsede: Accelerating scientific discovery, Computing in Science & Engineering 16 (05) (2014) 62–74. doi:10.1109/MCSE.2014.80.

[23] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy,

P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, R. Quick, The open science grid, Journal of Physics: Conference Series 78 (2007) 012057. doi:10.1088/1742-6596/78/1/012057.
URL https://doi.org/10.1088%2F1742-6596%2F78%2F1%2F012057

[24] D. Gunter, E. Deelman, T. Samak, C. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, K. Vahi, Online workflow management and performance analysis with stampede, in: 7th International Conference on Network and Service Management (CNSM-2011), 2011.

[25] Advanced Message Queuing Protocol (AMQP), https://www.amqp.org/.

[26] www.globus.org, globus, https://www.globus.org (2018 (accessed July 3, 2018)).

[27] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, S. Tuecke, Software as a service for data scientists, Communications of the ACM 55 (2) (2012) 81–88.

[28] Z. Liu, R. Kettimuthu, I. Foster, N. S. Rao, Cross-geography scientific data transfer trends and user behavior patterns, in: 27th ACM Symposium on High-Performance Parallel and Distributed Computing, HPDC '18, ACM, New York, NY, USA, 2018. doi:10.1145/3208040.3208053.
URL http://doi.acm.org/10.1145/3208040.3208053

[29] Z. Liu, P. Balaprakash, R. Kettimuthu, I. Foster, Explaining wide area data transfer performance, in: 26th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '17, ACM, New York, NY, USA, 2017, pp. 167–178. doi:10.1145/3078597.3078605.
URL http://doi.acm.org/10.1145/3078597.3078605

[30] R. Kettimuthu, Z. Liu, D. Wheeler, I. Foster, K. Heitmann, F. Cappello, Transferring a petabyte in a day, Future Generation Computer Systems 88 (2018) 191 – 198. doi:https://doi.org/10.1016/j.future.2018.05.051.
URL https://doi.org/10.1016/j.future.2018.05.051

[31] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, N. J. Wright, Modular hpc i/o characterization with darshan, in: Extreme-Scale Programming Tools (ESPT), Workshop on, IEEE, 2016, pp. 9–17.

[32] POSIX Standard, https://pubs.opengroup.org/onlinepubs/9699919799.

[33] HDF5, https://portal.hdfgroup.org/display/HDF5/HDF5.

[34] Jianwei Li, Wei-keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, Parallel netcdf: A high-performance scientific i/o interface, in: SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, 2003, pp. 39–39. doi:10.1109/SC.2003.10053.

[35] Tstat, TCP STatistic and Analysis Tool, http://tstat.tlc.polito.it.

[36] G. Juve, B. Tovar, R. Ferreira da Silva, D. Krol, D. Thain, E. Deelman, W. Allcock, M. Livny, Practical resource monitoring for robust high throughput computing, in: Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications, 2015. doi:10.1109/CLUSTER.2015.115.

[37] P. Gaikwad, A. Mandal, P. Ruth, G. Juve, D. Król, E. Deelman, Anomaly detection for scientific workflow applications on networked clouds, in: High Performance Computing & Simulation (HPCS), 2016 International Conference on, IEEE, 2016, pp. 645–652.

[38] R. Ferreira da Silva, M. Rynge, G. Juve, I. Sfiligoi, E. Deelman, J. Letts, F. Wurthwein, M. Livny, Characterizing a high throughput computing workload: The compact muon solenoid (CMS) experiment at LHC, Procedia Computer Science, International Conference On Computational Science, ICCS 2015 Computational Science at the Gates of Nature 51 (2015) 39–48. doi:10.1016/j.procs.2015.05.190.

[39] R. Ferreira da Silva, G. Juve, M. Rynge, E. Deelman, M. Livny, Online task resource consumption prediction for scientific workflows, Parallel Processing Letters 25 (3). doi:10.1142/S0129626415410030.

[40] Pivotal, Rabbitmq, https://www.rabbitmq.com/.

[41] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The Globus striped GridFTP framework and server, in: ACM/IEEE Conference on Supercomputing, SC '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 54–. doi:10.1109/SC.2005.72.
URL https://doi.org/10.1109/SC.2005.72

[42] Amazon, Amazon S3, https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html.

[43] D. Weitzel, M. Zvada, I. Vukotic, R. Gardner, B. Bockelman, M. Rynge, E. F. Hernandez, B. Lin, M. Selmeci, Stashcache: A distributed caching federation for the open science gridarXiv:arXiv:1905.06911, doi:10.1145/3332186.3332212.

[44] Z. Liu, R. Kettimuthu, I. Foster, P. H. Beckman, Towards a smart data transfer node, Future Generation Computer Systems (2018) 10.

[45] Scitech, Panorama kibana plugin, https://github.com/Panorama360/panorama-kibana-plugin (2018).

[46] SciTech, Pegasus panorama, https://github.com/pegasus-isi/pegasus/tree/panorama.

[47] Docker Inc., Docker, https://docs.docker.com/.

[48] Docker Inc., Docker Compose, https://docs.docker.com/compose/.

[49] SciTech, Panorama architecture backend, https://github.com/Panorama360/data-collection-arch (2019).

[50] T. Mason, D. Abernathy, I. Anderson, J. Ankner, T. Egami, G. Ehlers, A. Ekkebus, G. Granroth, M. Hagen, K. Herwig, et al., The spallation neutron source in oak ridge: A powerful tool for materials research, Physica B: Condensed Matter 385 (2006) 955–960.

[51] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, K. Schulten, Scalable molecular dynamics with namd on the ibm blue gene/l system, IBM Journal of Research and Development 26 (1.2) (2008) 1781–1802. doi:10.1147/rd.521.0177.

[52] R. Salomon-Ferrer, D. A. Case, R. C. Walker, An overview of the amber biomolecular simulation package, Wiley Interdisciplinary Reviews: Computational Molecular Science 3 (2) (2013) 198–210.

[53] B. Lindner, J. C. Smith, Sassena—x-ray and neutron scattering calculated from molecular dynamics trajectories using massively parallel computers, Computer Physics Communications 183 (7) (2012) 1491–1501.

[54] O. Arnold, J.-C. Bilheux, J. Borreguero, A. Buts, S. I. Campbell, L. Chapon, M. Doucet, N. Draper, R. F. Leal, M. Gigg, et al., Mantid—data analysis and visualization package for neutron scattering and μ sr experiments, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 764 (2014) 156–166.

[55] 1000 Genomes Project Consortium, A global reference for human genetic variation, Nature 526 (7571) (2012) 68–74.

[56] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, M. Atkinson, Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows, Future Generation Computer Systems 95 (2019) 615–628. doi:10.1016/j.future.2019.01.015.

[57] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, J. Chase, Exogeni: A multi-domain infrastructure-as-a-service testbed, in: Testbeds and Research Infrastructure. Development of Networks and Communities, Berlin, Heidelberg, 2012.

[58] Internet2, https://www.internet2.edu/.

[59] Energy Sciences Network (ESnet), https://www.es.net/.

[60] J. Chase, L. Grit, D. Irwin, V. Marupadi, P. Shivam, A. Yumerefendi, Beyond virtual data centers: Toward an open resource control architecture, in: International Conference on the Virtual Computing Initiative, 2007.

[61] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, P. Larroy, Linux advanced routing & traffic control, in: Ottawa Linux Symposium, Vol. 213, 2002.

[62] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, Y. Liu, Load balancing in data center networks: A survey, IEEE Communications Surveys & Tutorials.

[63] R. Ferreira da Silva, S. Callaghan, E. Deelman, On the use of burst buffers for accelerating data-intensive scientific workflows, in: 12th Workshop on Workflows in Support of Large-Scale Science (WORKS'17), 2017. doi:10.1145/3150994.3151000.

[64] R. Ferreira da Silva, S. Callaghan, T. M. A. Do, G. Papadimitriou, E. Deelman, Measuring the impact of burst buffers on data-intensive scientific workflows, Future Generation Computer Systems 101 (2019) 208–220. doi:10.1016/j.future.2019.06.016.

[65] A. Waterland, stress, POSIX workload generator (2013).