2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Toward an End-to-end Framework for Modeling, Monitoring and Anomaly Detection for Scientific Workflows

Anirban Mandal, Paul Ruth, Ilya Baldin
RENCI - UNC Chapel Hill
{*anirban,pruth,ibaldin*}*@renci.org*

Dariusz Król, Gideon Juve, Rajiv Mayani,
Rafael Ferreira da Silva, Ewa Deelman
USC Information Sciences Institute
{*darek,gideon,mayani,rafsilva,deelman*}*@isi.edu*

Jeremy Meredith, Jeffrey Vetter, Vickie Lynch,
Ben Mayer, James Wynne III
Oak Ridge National Laboratory
{*jsmeredith,vetter,lynchve,*
mayerbw,wynnejr}*@ornl.gov*

Mark Blanco, Chris Carothers,
Justin LaPre
Rensselaer Polytechnic Institute
blancm3@rpi.edu,
{*chrisc,laprej*}*@cs.rpi.edu*

Brian Tierney
LBL
bltierney@es.net

***Abstract*—Modern science is often conducted on large scale, distributed, heterogeneous and high-performance computing infrastructures. Increasingly, the scale and complexity of both the applications and the underlying execution platforms have been growing. Scientific workflows have emerged as a flexible representation to declaratively express complex applications with data and control dependences. However, it is extremely challenging for scientists to execute their science workflows in a reliable and scalable way due to a lack of understanding of expected and realistic behavior of complex scientific workflows on large scale and distributed HPC systems. This is exacerbated by failures and anomalies in large scale systems and applications, which makes detecting, analyzing and acting on anomaly events challenging. In this work, we present a prototype of an end-to-end system for modeling and diagnosing the runtime performance of complex scientific workflows. We interfaced the Pegasus workflow management system, Aspen performance modeling, monitoring and anomaly detection into an integrated framework that not only improves the understanding of complex scientific applications on large scale complex infrastructure, but also detects anomalies and supports adaptivity. We present a black box modeling tool, a comprehensive online monitoring system, and anomaly detection algorithms that employ the models and monitoring data to detect anomaly events. We present an evaluation of the system with a Spallation Neutron Source workflow as a driving use case.**

***Keywords*-scientific workflows, performance modeling, monitoring, anomaly detection**

I. Introduction

Modern computational and data science often involves processing and analyzing vast amounts of data through large scale simulations of underlying science phenomena. In addition, access to remote sensors, instruments, data-sets, high-performance computing resources, and demands for high-volume data flows are making modern scientific analysis highly dynamic, heterogeneous and distributed. This is evident in diverse fields of science including astronomy, bioinformatics, physics, and climate modeling among many others.

Not only are applications growing in scale and complexity, the distributed and high-performance computing infrastructure required to support science applications is increasingly diverse in scale and complexity — DOE Leadership Computing Facilities, OSG [1], XSEDE [2], cloud infrastructures [3], [4] and national and regional network transit providers like ESnet [5] and Internet2 [6]. It is extremely challenging for scientists to navigate the complexity of applications and infrastructures to execute their computational campaigns in a

978-1-5090-3682-0/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.202

reliable and scalable way. This, in part, stems from a lack of understanding of expected and realistic behavior of complex scientific workflows on large scale and distributed HPC systems. Furthermore, there are bound to be failures and anomalies in large scale systems and applications, and detecting, analyzing the root causes for, and acting on anomaly events are challenging.

In order to address some of the above challenges, the Panorama project [7] aims to further our understanding of the behavior of scientific workflows as they are executing in large scale, heterogeneous environments by modeling and diagnosing the run-time performance of complex scientific workflows. We are developing tools that analyze the workflow and that develop models of expected behavior given a particular computing environment, such as an HPC system, clusters distributed over wide area networks, or clouds. We are also investigating the use of analytical models for resource provisioning, scheduling and data management decisions. This is coupled with correlating real-time monitoring of application and infrastructure to verify application behavior, to detect and diagnose anomalies, and to support adaptivity.

In this paper, we present our initial successes in developing analytical models that can predict the behavior of complex, data-intensive, scientific workflows executing on large-scale infrastructures. In particular, we describe a black-box modeling approach for generating informed thresholds for various performance metrics using the Aspen analytical performance modeling language and suite of tools [8] (Section II). We also present the design and implementation of a comprehensive performance monitoring framework (Section III) for dynamically monitoring various workflow, application and infrastructure metrics, and how the monitoring data can be used to build initial black box models for performance prediction and for anomaly detection (Section IV) during workflow execution. The paper makes the case for an automated, end-to-end framework (Section VI) that ties together workflow planning and management using Pegasus [9], Aspen performance modeling, online monitoring, and anomaly detection and notification for improving the overall performance of complex scientific workflows on current and future generation architectures. We present a prototype system that used a Spallation Neutron Source workflow (Section V) as the driving use case.

II. Analytical Modeling with Aspen Black-box Modeling Tool

To generate more informed thresholds for the online monitoring and anomaly detection, we turn to the Aspen analytical performance modeling language and suite of tools [8].

There are myriad ways of generating application models for Aspen, including automated parsing of source code [10] and through manual efforts from those knowledgeable about the application in question. In the case of modeling workflows, our system may be exposed to applications for which no model has yet been created, and it must be able to generate informed thresholds for anomaly detection even in the absence of more detailed descriptions of these applications. For this purpose, we created a technique to use Aspen for black-box application model generation.

A. Black-box Modeling for Recorded Data

Suppose we have obtained information about runtimes for some application, running on the CPU cores of a known machine, at two different problem sizes (n). This information is seen in the following table:

n	machine	socket	runtime
100	machine.aspen	cpu	11
500	machine.aspen	cpu	15

A simple solution to generating an analytical performance model for this application would to perform, say, a linear regression on the runtime. In particular, we could predict that $runtime = 10 + \frac{n}{100}$. This is very easy, but provides minimal information about the application itself, and provides no ability to extrapolate to other machines as will likely be needed by a workflow system.

A better solution is to use some known aspect of the machine to transition away from a model based on runtime alone. For example, as the known

abstract machine model for our system has a clockspeed for the CPUs, such as 1 GHz, and we know that it can perform one floating point operation per cycle, then we can refine our prediction to $flops = 10^{10} + n \times 10^{7}$. This prediction is now *machine-independent*, and we have a performance model based solely on application parameters and application resource usage. In other words, we now have an *application model* which can be generate performance predictions for the current machine as well as unknown future machines.

Our system takes this approach one step further and can automatically generate an application model based on other resource parameters such as floating point operations, bytes loaded and stored from memory, and messages communicated between tasks – i.e., from any application resource usage which can be described for an abstract machine model.

Similar work has been done in this area. The approach taken in [11] utilized runtimes and flops, though lost some accuracy by ignoring threshold factors such as I/O, contention, and network latency. A later approach in [12] added a genetic programming error correction procedure to increase the overall accuracy. The approach we describe here allows for greater abstraction, flexibility, and portability by supporting any resource representable in template application models and abstract machine models within the Aspen framework.

B. Aspen Black-box Modeling Tool

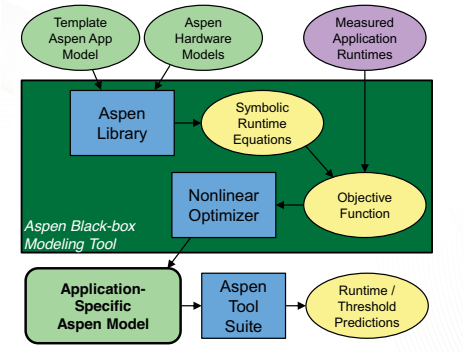

Figure 1. Aspen black box modeling tool.

The black-box modeling tool in Aspen uses the following approach, as shown in Figure 1:

- The user chooses a template Aspen model or creates their own. This template model contains free parameters used to fit the data.
- The user feeds this template model and recorded runtime data into the Aspen black-box modeling tool.
- Aspen converts the templated Aspen model into symbolic runtime equations for each machine listed in the input runtime data.
- The modeling tool creates an objective function which returns the error between the runtime predictions and the recorded data for a given set of parameters. This is currently done via least-squares fitting.
- An optimizer (such as *http://ab-initio.mit.edu/nlopt/*) solves for the free parameters to minimize the error of the objective function.
- The tool outputs a concrete Aspen application model which combines the input template model and the best solution to the free parameters.

An example input template Aspen application model is as follows:

```
model NAMD_Template {
  param nAtoms     = 1e6     // application parameters
  param nTimeSteps = 100  // (defined in the input file)
  param c = 1 in 1 .. 1e18   // solve for these parameters
  param d = 1 in 1 .. 1e18   // (within the given ranges)
  kernel main  // application behavior: execution and control flow
  { iterate [nTimeSteps] {
      execute {
        loads [c * nAtoms^2]
        flops [d * nAtoms]
  }}}
}
```

Here, we see problem parameters like `nAtoms` and `nTimeSteps`. These values can be defined in the input data file for each recorded runtime. We also see free parameters `c` and `d`; the modeling tool recognizes the allowable ranges on these parameters and uses them as constraints during the optimization phase. Finally, we see the control flow and resource usage (`flops` and `loads`); this is a simple example but shows how some minimal knowledge about the application (e.g. resource usage scaling versus problem size, and a linear effect of number of time steps) can be used to generate a more informed template file for

modeling. The model output by the tool, now in concrete form with problem-specific parameters, is as follows:

```
model NAMD_Equilibrate {
  param nAtoms     = 1e6  // NAMD input parameters
  param nTimeSteps = 100
  param c = 402.1   // calculation-specific constants
  param d = 10.95
  kernel main  // NAMD application behavior
  { iterate [nTimeSteps] {
      execute {
        loads [c * nAtoms^2]
        flops [d * nAtoms]
  }}}
}
```

C. Interaction with CODES

CODES [13] is an HPC storage and network simulation framework built on the ROSS [14] parallel simulation framework which implemented the Time Warp parallel discrete-event simulation protocol using reverse computation [15]. ROSS has been shown to efficiently scale to nearly 2 million cores on the largest supercomputers available to date [14]. These scaling results have enabled CODES to model Dragonfly, Slimfly and Torus topologies with over 1 million nodes.

These large-scale network models can be linked to Aspen compute node models to provide a combined, overall picture of an applications performance. In the current integration, CODES drives Aspen to report computational kernel time information. That information is used by CODES to both delay and generate application specific communication patterns into the simulated network. When executing together, the Aspen/CODES integrated model reports no performance loss over just CODES along when executing a large-scale Dragonfly network model using 1024 Blue Gene/Q nodes with 8,192 MPI ranks.

III. Monitoring

Workflow modeling and analysis methods require detailed event traces and online monitoring data in order to build models of system behavior and perform anomaly detection and diagnosis as the workflow is running. The Panorama project has developed a sophisticated set of tools and infrastructure for collecting traces and monitoring data for workflows.

A. Workflow and application monitoring

The `monitord` component of Pegasus collects, aggregates and publishes all the monitoring data produced by the workflow. This includes high-level information on the state of the workflow generated by the workflow execution engine (DAGMan) and the workflow scheduler (HTCondor), as well as low-level information published by job monitoring tools.

Job-level monitoring is performed by Kickstart [16], a monitoring and tracing tool used to launch computation and data management jobs to collect information about the behavior of the jobs and their execution environment. A Kickstart process forks application processes on the compute node and uses its position as the parent process to inspect the behavior of the application. Kickstart writes trace data to a file for offline analysis, and reports real-time monitoring data to monitord.

As part of the DOE dV/dt project [17], we added functionality to Kickstart to automatically capture resource usage metrics of workflow jobs [18]. This functionality uses operating system monitoring facilities such as `procfs` and `getrusage()` as well as library call interposition to collect fine-grained profile data that includes process I/O, file accesses, runtime, memory usage, and CPU utilization.

Library call interposition is used to implement monitoring functionality that requires access to the internal state of an application process. It is implemented by a component called `libinterpose`, which uses `LD_PRELOAD` to intercept calls to POSIX functions for file and network I/O and threads, as well as performing monitoring activities at process start and exit, such as starting monitoring threads, activating CPU counters, and reporting final performance metrics.

We extended Kickstart and libinterpose to collect CPU performance counters using the PAPI library [19]. libinterpose enables the CPU counters when the process starts, and reports their values periodically to Kickstart. This includes counters for the number of floating-point operations, instructions, loads, stores, and cache misses exe-

cuted by the application. We expect this information to be very useful in modeling because it tells us more about the computational requirements of the application that is independent of the execution hardware. Previously, our modeling was focused on metrics such as runtime, which are a function of both the computation requirements and the hardware capability. By measuring the computation in terms of loads, stores, and operations, we expect to be able to construct machine-independent models of the application using Aspen (Section II-A) that have the potential to be evaluated against different execution hardware.

We also extended Kickstart to support MPI jobs. Previously Kickstart was not able to monitor MPI processes running on multiple compute nodes without running a Kickstart process for each MPI rank. Using libinterpose, however, there can be one Kickstart process that invokes `mpiexec` (or equivalent) to launch the MPI job, and libinterpose can attach to each MPI process and report results back to the Kickstart process. To facilitate this, libinterpose sends MPI rank information along with monitoring data to Kickstart, and this data is aggregated across all MPI ranks to produce one, unified time series for the entire MPI job.

B. Infrastructure monitoring

We are also collecting infrastructure-level monitoring data for network, storage system, and host performance. Our current tools collect monitoring data from the infrastructure with particular emphasis on I/O and network performance, which are critical for many workflows. The goals are to gather monitoring data to help build initial models. Workflow- and job-level monitoring is correlated with infrastructure-level monitoring to identify anomalies and their potential sources. We are using a combination of standard system monitoring tools, including `sar`, and automated scripts, as well as active monitoring infrastructure such as perfSONAR [20] and fio [21]. We are using racks on the ExoGENI [3] testbed as a controlled environment to collect monitoring data, both application-level monitoring data and infrastructure-level monitoring data, because it offers minimal performance interference, which is extremely important to build and validate initial models.

During workflow execution, we launch automated monitoring scripts as a part of workflow launch bootstrap to observe various OS level performance metrics on the nodes. The monitoring scripts use the `sar` tool from the "sysstat" [22] linux utilities for online monitoring of I/O and network. For example, if the workflow applications are using a shared filesystem like NFS, we monitor the I/O performance of the node that acts as the NFS server. In this case, we monitor number of observed read/write requests and observed read/write bandwidth on a node using the '-b' option in `sar`. In the network case, we observe receive/transfer bandwidth and the number of packets received/transferred using the '-n' option in `sar` on the data-plane network interface, which is used for application data movement. All the above infrastructure monitoring data is collected every five seconds and stored as a time series.

In addition, we have created a tool for actively monitoring block storage I/O performance, which involves running periodic tests on the ExoGENI racks. The tool works by probing each ExoGENI site and submitting a resource request that provisions a machine, runs a suite of I/O performance benchmarks, and stores the performance results as a time series. Our tool uses fio to simulate and benchmark I/O load. fio is a flexible tool for benchmarking I/O performance and generating simulated I/O load. We have configured fio to simulate several representative patterns of I/O including reading and writing of sequential and random I/O blocks. For each pattern of I/O in the benchmark suite, we are collecting several metrics that include read/write bandwidth, latency, and latency percentiles. For active network monitoring, we created a perfSONAR image based on Docker for the ExoGENI infrastructure testbed. We updated the base image with client tools that can push perfSONAR network monitoring data to our monitoring database. We have provisioned resources between different pairs of racks, and ran

perfSONAR bwctl tests with appropriate parameters to bwctl.

IV. Anomaly Detection

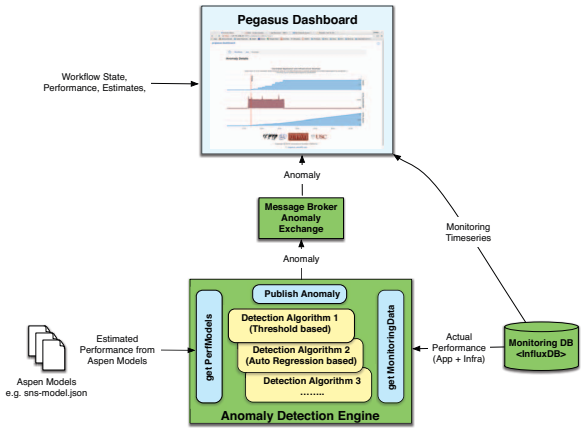

Figure 2. Anomaly detection components.

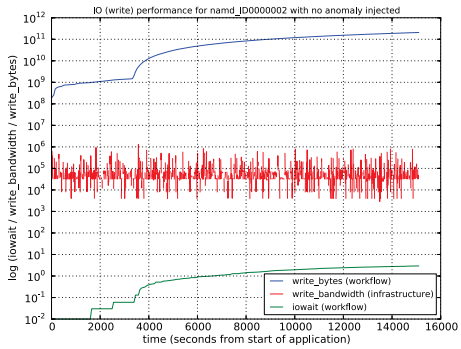

Figure 3. AR based detection example (training series).

Coupling online monitoring data from workflow application and infrastructure with the Aspen performance models gives us a powerful tool to detect anomalies during workflow executions. An accurate attribution of the anomalies to the observed workflow performance is critical for fixing the underlying problem, or adapting the system. In this section, we describe the various components of the anomaly detection engine and its interactions with the rest of the system (Figure 2).

The "getMonitoringData" component interacts with the online monitoring data to obtain the different time series for relevant performance metrics. The "getPerfModel" component is designed to interact with the Aspen performance modeling system to obtain the estimated performance of workflow applications from the Aspen models. In the current implementation, it statically reads the generated thresholds from the Aspen black box modeling tool. In future, we plan a tighter integration with the modeling system. The "Publish Anomaly" component is responsible for generating anomaly events and publishing the anomaly messages to an AMQP message broker. The anomaly messages are then consumed by the Pegasus dashboard to enable users to see the details of the anomalies and the corresponding time series data, which is useful for attribution of anomalies.

The heart of the anomaly detection engine is a suite of detection algorithms, which are pluggable modules. These detection algorithms can be coupled in different ways to detect and correlate anomalies. Three types of detection algorithms are currently implemented in the system. The first is a simple threshold based detection algorithm that does continuous diffs between Aspen generated thresholds and online measurements for application specific metrics. This can be coupled with a "MovingAverage" (MA) based detection algorithm for analyzing time-series data for infrastructure related metrics. We have also developed "AutoRegression" (AR) based algorithms to detect anomalies by analyzing time-series data for application related metrics. An AR model is first developed from non-anomalous runs, which includes the AR coefficients and the degree of AR that fits best for a metric. When ran against online monitoring data, the AR model is used to predict the errors that become significant in presence of anomalies.

We now present an example of the AR based detection technique. The first graph (Figure 3) shows the time series plots of two application metrics (write_bytes and iowait) related to I/O performance of the NAMD application from the SNS workflow (Section V) and one infrastructure related metric (write_bandwidth) related to

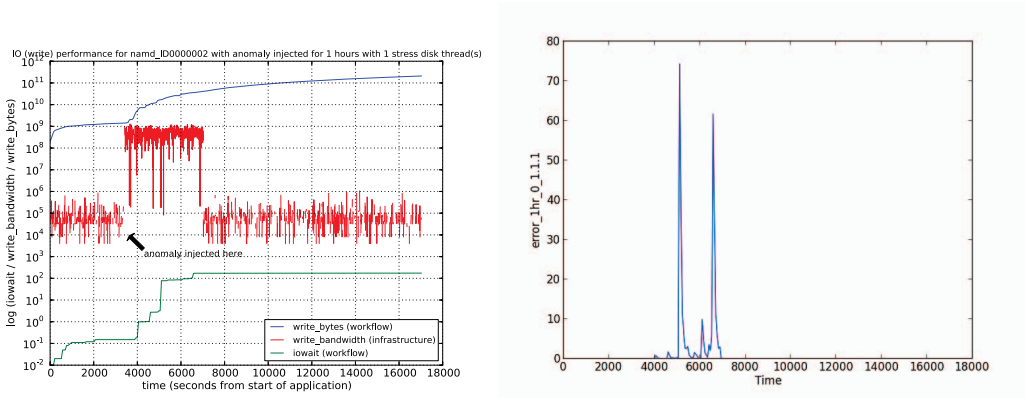

Figure 4. AR based detection example (anomalous series).

observed I/O on the node for a non-anomalous run. The X-axis represents time since the start of the application. In this example, the AR model is calculated for the iowait metric. Figure 4 shows the time series plots of the three metrics for an anomalous run where a write anomaly was introduced one hour into the run for an hour by utilizing the `stress` benchmark[1] on the node that acts as the NFS server. The `stress` benchmark is a workload generator for POSIX systems, and provides the capability to impose a configurable amount of CPU, memory, I/O, and disk stress on the system. In this case, we employed `stress` to inject I/O anomaly by only running disk stress threads. We observe the anomaly showing up as a step in the infrastructure time series plot. The second subfigure plots the errors from the AR model for iowait for each point of time during execution of NAMD. We observe that the errors are maximum in the region where anomaly existed and they cease to exist when anomaly is removed. We can use the AR model errors to trigger observed anomalies in an online fashion.

V. SNS Workflow

We have developed several use-case applications to evaluate our workflow modeling, monitoring and anomaly detection system. The research described in this paper uses the Spallation Neutron Source (SNS) Workflow as a test case.

[1]http://people.seas.harvard.edu/~apw/stress/

The Spallation Neutron Source (SNS) [23] is a research facility at the Department of Energy's Oak Ridge National Laboratory that uses pulsed neutron beams to investigate the properties of materials for scientific and industrial research. SNS uses a particle accelerator to impact a mercury-filled target with a stream of protons, generating neutrons by the process of spallation. These neutrons are directed toward a sample, causing some neutrons to scatter. The scattering events are collected by array of detectors and distilled into different science products depending on the experiment. This reduced data is then analyzed and compared to materials simulations to extract scientific information about the material.

In collaboration with the Center for Accelerating Materials Modeling (CAMM), we created a workflow that executes an ensemble of molecular dynamics and neutron scattering simulations to fit model parameter values to experimental results. This parameter refinement workflow has been used to investigate temperature and hydrogen charge parameters for models of water molecules.

The workflow executes one job to unpack a reference database along with 5 jobs for each set of parameter values. Each parameter value is fed into a series of parallel molecular dynamics simulations using NAMD [24]. The output from the MD simulations has the global translation and rotation removed using AMBER's [25] cpptraj utility [26] and is passed to Sassena [27] for the

calculation of coherent and incoherent neutron scattering intensities. The final outputs of the workflow are transferred to the user's desktop and loaded into Mantid [28] for analysis and visualization.

The production version of this workflow has been run on the Hopper supercomputer at NERSC and is in the process of being ported to the Titan supercomputer at OLCF.

VI. End-to-end System

The end-to-end workflow modeling, monitoring, and anomaly detection system is illustrated in Figure 5.

The workflow is planned and executed using the Pegasus Workflow Management System. Pegasus converts an abstract workflow description into an executable, directed acyclic graph (DAG) of jobs that is managed by the DAGMan [29] workflow engine. DAGMan submits the jobs in the DAG to HTCondor [30], which executes them on the distributed infrastructure. As the workflow is running, the Pegasus `monitord` service collects workflow monitoring data, and updates the state of the workflow in the Pegasus database.

Application monitoring is implemented by wrapping each job in the workflow with Kickstart. Kickstart performs two functions: it collects trace data for offline analysis and modeling, and it monitors the job and reports real-time performance metrics via a RabbitMQ[2] message broker. The `monitord` service aggregates monitoring data for a job from RabbitMQ, updates the current metrics for the job in the Pegasus database, and publishes job performance metrics to an InfluxDB[3] time series database.

Infrastructure monitoring is implemented using the infrastructure monitoring tools as described in Section III-B. These tools continuously collect monitoring metrics for the hosts, storage systems and networks used by the workflow and store them as time series in InfluxDB.

[2]https://www.rabbitmq.com
[3]https://influxdata.com

Anomaly detection tools use models to predict application performance, and compare the predicted performance to the observed performance from InfluxDB. When an anomaly is detected, a message is published via RabbitMQ that describes the details of the anomaly. These anomaly messages are collected and stored in the Pegasus database by `monitord`.

Monitoring and anomaly data are displayed in a web-based user interface called the Pegasus Dashboard. Users can select a job and see current values for all of the performance metrics collected by the online monitoring infrastructure, as well as time series plots of the data for the job from InfluxDB. The Dashboard also displays anomalies that were detected by the anomaly detection tools. The Dashboard reads these anomalies from the Pegasus database and displays them with along with time series plots from InfluxDB that help to illustrate and explain the cause of the anomaly.

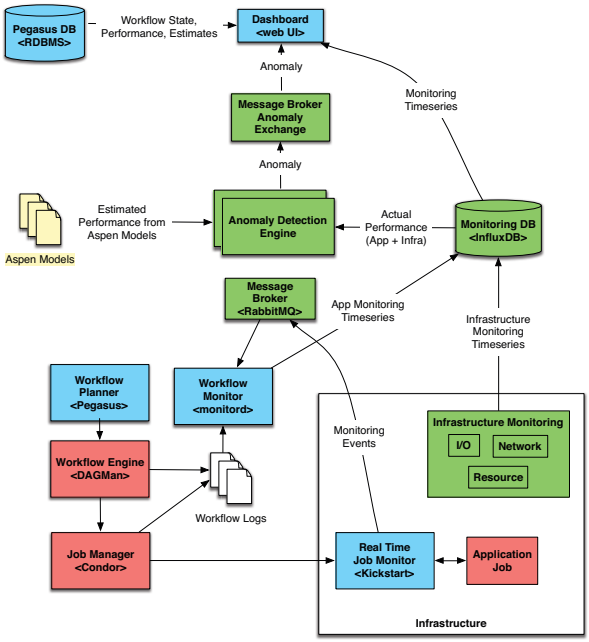

Figure 5. End-to-end workflow modeling, monitoring and anomaly detection system. Red indicates existing components, blue indicates modified components, and green indicates new components.

We deployed the SNS workflow application on an ExoGENI [3] rack located at the Pittsburgh Supercomputing Center. We instantiated a virtualized

HTCondor system capable of running MPI simulations utilizing 100 cores and ran the SNS workflow application using Pegasus. As discussed earlier, we collected online monitoring data from application and infrastructure during workflow execution. We introduced an I/O anomaly using the `stress` tool. This anomaly was designed to simulate competition with other jobs for I/O resources, which is a common source of job performance variability on production HPC systems. The Aspen model thresholds were available to the anomaly detection engine, and we ran the MA- and threshold-based detection algorithms. The I/O anomaly appears in the application monitoring as a spike in the `iowait` metric, which measures the amount of time that the process was blocked waiting on I/O. It was detected by the system and shown on the Pegasus dashboard with the corresponding relevant time series, as shown in Figure 6.

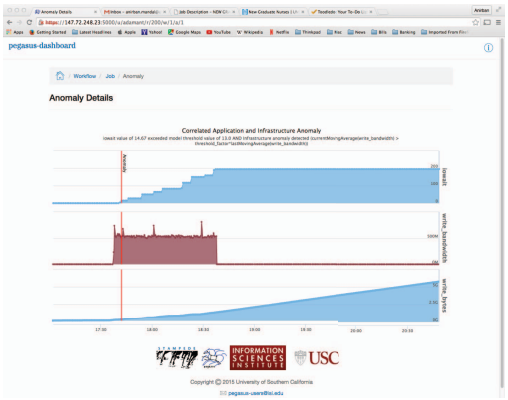

Figure 6. Pegasus dashboard anomaly notification.

VII. Conclusion

We presented a prototype of an integrated, end-to-end system that models, executes, monitors, and detects anomalies for complex scientific workflows. The system has been applied and evaluated in the context of executing SNS workflows on the ExoGENI testbed. It is a foundational framework for our future work on developing more complex predictive models, anomaly detection algorithms, workflow adaptation mechanisms, and adaptive resource provisioning for workflows executing on distributed, heterogeneous and HPC systems.

Acknowledgment

This work was funded by DOE under contract #DE-SC0012636, "Panorama - Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows". The development of the neutron scattering simulation workflow was supported by the U.S. Department of Energy (DOE), Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division. The use of Oak Ridge National Laboratory's Spallation Neutron Source was sponsored by the Scientific User Facilities Division, Office of Basic Energy Sciences.

References

[1] "Open Science Grid," http://www.opensciencegrid.org.

[2] "Extreme Science and Engineering Discovery Environment," http://www.xsede.org.

[3] I. Baldine, Y. Xin *et al.*, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, 2012, pp. 97–113.

[4] "CloudLab," http://cloudlab.us.

[5] "ESnet," http://www.es.net.

[6] "Internet2," http://www.internet2.edu.

[7] E. Deelman, C. Carothers, A. Mandal, B. Tierney, J. S. Vetter, I. Baldin, C. Castillo, G. Juve, D. Król, V. Lynch, B. Mayer, J. Meredith, T. Proffen, P. Ruth, and R. Ferreira da Silva, "PANORAMA: An approach to performance modeling and diagnosis of extreme scale workflows," *International Journal of High Performance Computing Applications*, vol. to appear, 2015.

[8] K. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," in *SC12: ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2012.

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

[10] S. Lee, J. S. Meredith, and J. S. Vetter, "COMPASS: A framework for automated performance modeling and prediction," in *ACM International Conference on Supercomputing (ICS)*. Newport Beach, California: ACM, 2015.

[11] G. Mahinthakumar, M. Sayeed, J. Blondin, P. Worley, A. Mezzacappa, and R. Hix, "Performance evaluation and modeling of a parallel astrophysics application," in *Proceedings of the High Performance Computing Symposium*, J. Meyer, Ed., 2004, pp. 27–33.

[12] K. Raghavachar, G. Mahinthakumar, P. Worley, E. Zechman, and R. Ranjithan, "Parallel performance modeling using a genetic programming-based error correction procedure," *SIMULATION*, vol. 83, no. 7, pp. 515–527, 2007.

[13] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "A case study in using massively parallel simulation for extreme-scale torus network codesign," in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS '14. New York, NY, USA: ACM, 2014, pp. 27–38.

[14] P. D. Barnes, Jr., C. D. Carothers, D. R. Jefferson, and J. M. LaPre, "Warp Speed: Executing Time Warp on 1,966,080 Cores," in *Proc. 2013 ACM SIGSIM Conf. Principles of Advanced Discrete Simulation*, ser. PADS '13, Montreal, Canada, 19–22 May 2013, pp. 327–336.

[15] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. Model. Comput. Simul.*, vol. 9, no. 3, pp. 224–253, Jul. 1999.

[16] J. S. Vockler, G. Mehta, Y. Zhao, E. Deelman, and M. Wilde, "Kickstarting remote applications," in *International Workshop on Grid Computing Environments*, 2007.

[17] "dv/dt: Accelerating the rate of progress towards extreme scale collaborative science," https://sites.google.com/site/acceleratingexascale/.

[18] G. Juve, B. Tovar, R. Ferreira da Silva, D. Król, D. Thain, E. Deelman, W. Allcock, and M. Livny, "Practical resource monitoring for robust high throughput computing," in *2nd Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications*, ser. HPCMASPA'15, 2015, pp. 650–657.

[19] K. London, S. Moore, P. Mucci, K. Seymour, and R. Luczak, "The papi cross-platform interface to hardware performance counters," in *Department of Defense Users? Group Conference Proceedings, Biloxi, Mississippi*, vol. 3, no. 4, 2001.

[20] B. Tierney, J. Metzger, J. Boote, E. Boyd, A. Brown, R. Carlson, M. Zekauskas, J. Zurawski, M. Swany, and M. Grigoriev, "perfsonar: Instantiating a global network measurement framework," *SOSP Wksp. Real Overlays and Distrib. Sys*, 2009.

[21] "fio - Flexible I/O Tester," https://github.com/axboe/fio.

[22] "sysstat," http://sebastien.godard.pagesperso-orange.fr.

[23] T. Mason, D. Abernathy *et al.*, "The spallation neutron source in oak ridge: A powerful tool for materials research," *Physica B: Condensed Matter*, vol. 385, pp. 955–960, 2006.

[24] J. C. Phillips, R. Braun *et al.*, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[25] D. Case, V. Babin *et al.*, "AMBER 14, university of california, san francisco," 2014.

[26] D. R. Roe and I. Thomas E. Cheatham, "Ptraj and cpptraj: Software for processing and analysis of molecular dynamics trajectory data," *Journal of Chemical Theory and Computation*, vol. 9, no. 7, pp. 3084–3095, 2013.

[27] B. Lindner and J. Smith, "Sassena: X-ray and neutron scattering calculated from molecular dynamics trajectories using massively parallel computers," *Computer Physics Communications*, vol. 183, no. 7, pp. 1491–1501, 2012.

[28] O. Arnold, J. C. Bilheux *et al.*, "Mantid: Data analysis and visualization package for neutron scattering and sr experiments," *Nuclear Instruments and Methods in Physics Research Section A*, vol. 764, pp. 156–166, November 2014.

[29] "DAGMan," http://research.cs.wisc.edu/htcondor/dagman/dagman.html.

[30] "HTCondor," http://research.cs.wisc.edu/htcondor.