Anomaly Detection for Scientific Workflow Applications on Networked Clouds

Prathamesh Gaikwad, Anirban Mandal, Paul Ruth
RENCI - UNC Chapel Hill
pratham@cs.unc.edu
{anirban, pruth}@renci.org

Gideon Juve, Dariusz Król, Ewa Deelman
USC Information Sciences Institute
{gideon, darek, deelman}@isi.edu

***Abstract*—Recent advances in cloud technologies and on-demand network circuits have created an unprecedented opportunity to enable complex scientific workflow applications to run on dynamic, networked cloud infrastructure. However, it is extremely challenging to reliably execute these workflows on distributed clouds because performance anomalies and faults are frequent in these systems. Hence, accurate, automatic, proactive, online detection of anomalies is extremely important to pinpoint the time and source of the observed anomaly and to guide the adaptation of application and infrastructure. In this work, we present an anomaly detection algorithm that uses auto-regression (AR) based statistical methods on online monitoring time-series data to detect performance anomalies when scientific workflows and applications execute on networked cloud systems. We present a thorough evaluation of our auto-regression based anomaly detection approach by injecting artificial, competing loads into the system. Results show that our AR based detection algorithm can accurately detect performance anomalies for a variety of exemplar scientific workflows and applications.**

Keywords*—anomaly detection; scientific workflows; networked clouds; performance monitoring

I. Introduction

With the advent of pervasive virtualization, compute, network and storage infrastructure is becoming increasingly programmable at all layers, and end-to-end virtual environments can be provisioned dynamically in support of science applications. The recent advances in enabling on-demand network circuits, coupled with programmable cloud technologies [1], [2] create an unprecedented opportunity to enable complex data-intensive scientific applications to run on elastic *networked cloud* infrastructure. We refer to this model as Networked Infrastructure-as-a-Service (NIaaS).

Networked cloud infrastructures link distributed resources into connected arrangements, *slices*, targeted at solving a specific problem. This *slice* abstraction is central to providing mutually isolated pieces of networked virtual infrastructure, carved out from multiple cloud and network transit providers, and built to order for guest applications like scientific workflows (Figure 1). One such exemplar NIaaS system used in this work is ExoGENI [3] (Section II-A).

Scientific workflows are becoming a centerpiece of modern computational and data-intensive science. Scientific workflows have emerged as a flexible representation to declaratively express complex applications with data and control dependencies. The inherent elasticity present in scientific workflows, i.e. evolving resource needs as they execute, makes networked clouds an attractive platform for deploying and executing science workflows. Advanced virtualization technologies now make it possible to package execution environments in a way that science workflows can be highly portable, predictable, high-performance, and performance isolated.

However, it is extremely challenging for scientists to execute their science workflows in a reliable and scalable way. Performance degradation and faults are frequent while executing science workflows and applications on distributed cloud platforms. Failures and anomalies at all levels of the system - hardware infrastructure, system software, middleware, application and workflows make detecting, analyzing and acting on anomaly events challenging. Hence, automatic, proactive, online detection of anomalies is extremely important to take remedial actions during runtime so that there is minimal resource wastage and minimal effect on the workflow makespan. Accurate anomaly detection helps pinpointing the time and source of anomaly and thereby helps choosing the correct adaptation mechanism, both for the workflow application and the infrastructure.

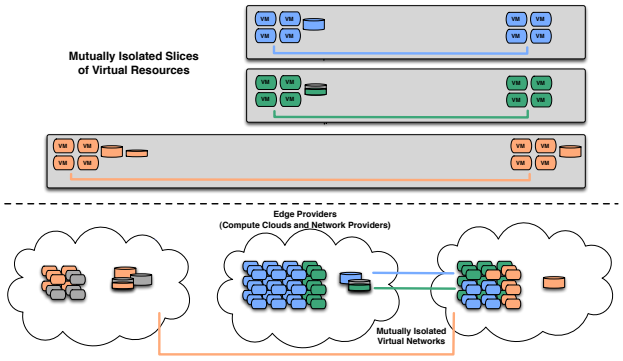

Fig. 1. Networked clouds.

In this work, we address the challenges of online detection of anomalies while executing scientific workflow applications on networked clouds. Using an integrated framework to collect online monitoring time-series data from application and infrastructure developed as part of the Panorama [4] project, we developed online detection algorithms to detect performance anomalies when science workflows and applications

are executed. The detection algorithms use regression based statistical methods to generate model parameters to fit the data and calculate error based on predicted and actual values to generate anomaly triggers. We present a thorough evaluation of our auto-regression (AR) based anomaly detection approach to detect performance anomalies for a variety of scientific workflows and applications. We execute several HPC applications and workflows on the ExoGENI networked cloud testbed to show the effectiveness of AR based time series analysis to detect anomalies when artificial, competing loads are injected into the system.

The paper is organized as follows. Section II provides background on ExoGENI, the Pegasus workflow management system, and deployment of the workflows on ExoGENI. Section III describes the monitoring framework and the AR based anomaly detection algorithm. Section IV presents an evaluation of the anomaly detection technique for several exemplar scientific workflows and applications. Section V presents related work and section VI concludes the paper.

II. Background

A. ExoGENI

ExoGENI [3] orchestrates a federation of independent cloud sites located across the US and circuit providers, like Internet2 and ESNet, through their native IaaS API interfaces. It is a widely distributed networked infrastructure-as-a-service (NIaaS) platform geared towards experimentation and computational tasks. Virtual compute, storage, and network resources are stitched together to form mutually isolated *"slices"* of infrastructure. Experimental slices can have complex topologies that span one or more ExoGENI sites. Users and applications can get resources from ExoGENI racks by submitting their slice requests using several command-line and GUI tools. ExoGENI employs sophisticated topology embedding algorithms that take advantage of semantic resource descriptions, and map slice topology requests to underlying infrastructure actions that instantiate the virtual infrastructure for guest applications.

The ExoGENI testbed, as shown in Figure 2, consists of cloud sites (*racks*) on more than 15 host campuses, linked with national research networks through programmable exchange points. Compute and storage resources are obtained from private clouds at the infrastructure's edge. Network resources are obtained from both edge providers and national fabrics using traditional VLAN-based switching and OpenFlow. ExoGENI uses the ORCA (Open Resource Control Architecture) [5] control framework software to offer a unified hosting platform for deeply networked, multi-domain cloud applications. The work in this paper uses ExoGENI as an exemplar NIaaS system.

B. Pegasus Workflow Management System

The Pegasus Workflow Management System [6] has been used by scientists in many different domains to execute large-scale computational workflows on a variety of cyberinfrastructure, ranging from local desktops to campus clusters, grids, and commercial and academic clouds. In this work, we have used Pegasus as a representative workflow management system leveraging NIaaS platforms to plan and execute workflows.

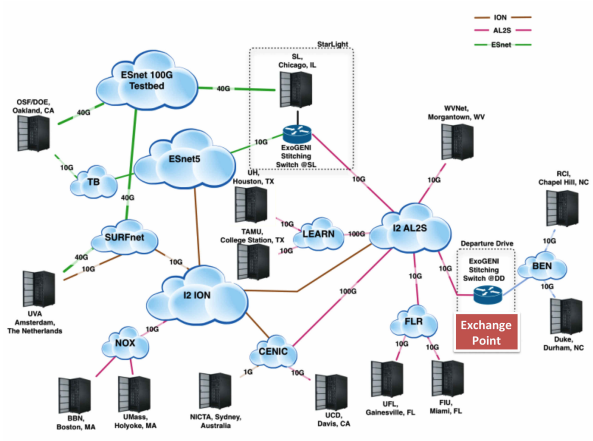

Fig. 2. ExoGENI testbed.

Pegasus focuses on scalable, reliable and efficient workflow execution on a wide range of systems. The key idea behind Pegasus is the separation of the workflow description from the description of the execution environment, which results in: 1) workflow portability, and 2) the ability for the workflow management system to make performance- and reliability-focused decisions at "planning time" and at "runtime". Pegasus pioneered the use of planning in scientific workflow systems [7]. The Pegasus planner takes a resource-independent workflow description, automatically locates the input data and computational resources necessary for workflow execution, applies performance optimizing transformations, maps the workflow onto the available execution resources, then reliably executes the plan. Pegasus leverages HTCondor [8] as the underlying execution framework for the workflow jobs.

C. Deploying Workflows on ExoGENI Testbed

We have used racks from the ExoGENI NIaaS platform as a controlled testbed environment to experiment with anomaly detection for workflows. Details on the hardware used in the racks can we found at [9]. Slices were provisioned from these racks by sending requests for virtual topologies consisting of a set of virtual machines (VM) connected via a broadcast link with a specified bandwidth, potentially spread across multiple racks. In this work, the Pegasus workflow management system, described above, was used to plan workflows that execute on HTCondor based systems.

The experiments performed in our work involved deploying a complete HTCondor site within a slice of ExoGENI resources by sending a request to ExoGENI to instantiate a virtual cluster customized for HTCondor. The HTCondor site includes one HTCondor scheduler/master (head node), and several HTCondor workers (HTCondor startd's). The head node and workers are deployed within virtual machines with a dedicated layer-2 network. The amount of bandwidth allocated to the network is configurable.

The VM images had pre-requisite software installed like HTCondor, Pegasus, including monitoring extensions described later. The ExoGENI postboot script feature was leveraged to start various HTCondor daemons on VM startup so that the HTCondor environment is ready as soon as the slice setup is complete. In addition, the various scientific workflows and applications were pre-installed as part of virtual machine images. In essence, the ExoGENI platform enables users to request tailored networked cloud infrastructure, which can then be used to rapidly deploy applications and science workflows using workflow management systems like Pegasus.

III. Anomaly Detection System

In this section, we present an overview of the monitoring and data collection architecture for obtaining online performance data for science workflows and applications running on the ExoGENI testbed, and how the collected data can be used for runtime detection of anomalies using statistical methods and algorithms. First, we present our monitoring framework.

A. Monitoring and Data Collection Framework

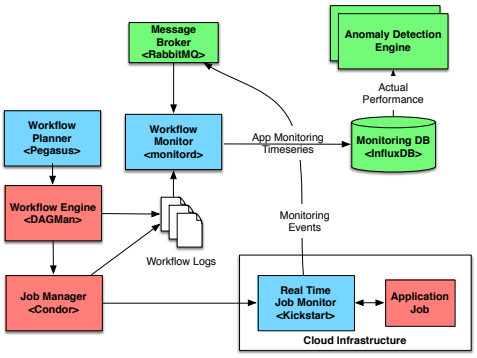

Fig. 3. Integrated workflow monitoring and anomaly detection system.

In this work, we leveraged the monitoring capabilities of the Pegasus Workflow Management System to collect online monitoring data. Figure 3 shows the overall architecture of the monitoring framework. Pegasus uses HTCondor DAGMan and the HTCondor scheduler as the underlying execution framework for launching workflow jobs on computational resources. The `monitord` service 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 [10], a monitoring and tracing tool used to launch computation and data management jobs, and 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 [11], we added functionality to Kickstart to automatically capture resource usage metrics for workflow jobs [12]. 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.

The `monitord` service publishes the resulting real-time monitoring data to a monitoring data store, which in our case is an InfluxDB [13] time series database. The Anomaly Detection Engine continuously queries this monitoring store, runs detection algorithms described in Section III-B on the online monitoring data, and sends triggers to the Pegasus Dashboard (not shown) to notify users of anomalies.

B. Auto-Regression (AR) Based Detection Algorithm

We now present details on the anomaly detection algorithm used by the detection engine described above. The architecture allows several detection algorithms to be pluggable in the detection engine. Analyzing time series data for detecting anomalies has been studied in the great detail in the literature (Section V). Among the several approaches proposed, we employed regression based statistical techniques on the monitoring time-series data.

Statistical techniques learn a model in the form of parameters, which describes the k^{th} observation in terms of previous $(k-1)$ observations. Using these learnt parameters we can also predict the k^{th} test value based on previous $(k-1)$ test observations. Using the predicted value and the actual value of test observation, error can be calculated at each test observation. If the error increases beyond a certain threshold then we say an anomaly has occurred. We use 2 techniques - Moving Average (MA) and Auto Regression, which differ in model parameters and previous observations chosen. In Moving Average, the first element is calculated by taking average of the initial fixed subset of series of numbers. To calculate the next element, the subset is modified by removing the first number of the series and including the next number following the last element of the original subset. We then calculate average of this new subset of numbers to get the next element. The process is repeated to get a plot of MA of the original series, where each point in MA is average of subset of original series of numbers. Moving average can be thought of as filter which ignores the instantaneous changes and captures the trend of the original series, since we always take average of subset of values. Although we have employed the MA

approach to detect anomalies in infrastructure time series data, i.e. monitoring data from the infrastructure, the focus of this paper is on detecting workflow application anomalies. We employed the Auto Regression technique for application time series data.

The AR model in terms of filter is sometimes referred to as recursive filter or infinite impulse response filter because it uses previous values of its own series to calculate the next value. In AR model of order p, the current term can be estimated by linear weighted sum of previous terms in series. The value of y(t) can be calculated using the following equation:

$$y(t) = \sum_{i=1}^{i=p} a_i y_{t-i} + \epsilon$$

where a_i are the AR coefficients, y is the series and p is order of the AR filter which is usually much less than the actual length of series y. The noise term in the above equation (ϵ) is assumed to be Gaussian white noise.

The problem statement in AR is to find the "best" values for a_i for a given series y of length N. AR model usually assumes series y is linear and stationary and has a zero mean. If the series y does have not have zero mean, we simply add a_0 in front of summation in the equation above.

The AR coefficients can be estimated using 3 different techniques 1) Yule-Walker method 2) Burg method and 3) Least squares method. The least squares method is based on the Yule-Walker method of estimating AR coefficients. In our detection framework, we have implemented the least squares method to estimate the AR coefficients. The equation to estimate AR coefficients using least square method for model of order p is given by:

$$\begin{pmatrix} c_{11} & c_{12} & ... & c_{1p} \\ c_{21} & c_{22} & ... & c_{2p} \\ . & . & ... & . \\ . & . & ... & . \\ . & . & ... & . \\ c_{p1} & c_{p2} & ... & c_{pp} \end{pmatrix} \begin{pmatrix} \hat{a_1} \\ \hat{a_2} \\ . \\ . \\ . \\ \hat{a_p} \end{pmatrix} = - \begin{pmatrix} c_{01} \\ c_{02} \\ . \\ . \\ . \\ c_{0p} \end{pmatrix}$$

where,

$$c_{ij} = 1/N - p \sum_{t=p+1}^{t=N} y_{t-i} y_{t-j}$$

and c_{ij} is an estimate of the auto-covariance function for delay (i - j) between y_{t-i} and y_{t-j} and N is the length of series. We solve the above set of equations to obtain $\hat{a_1}$, $\hat{a_1}$ .. $\hat{a_p}$, which are estimated coefficients of AR(p) model.

To calculate the optimal value of *order* of model (p) we must draw a plot of Root Mean Square (RMS) error vs. order of model. When we gradually increase the order, we see RMS error keeps on reducing until there is no significant effect of increasing order on the RMS error. After a certain order, the RMS error is almost constant and plot of RMS error vs. order(p) looks like flat line parallel to the X-axis. The optimal value of order(p) is somewhere near where line starts to flatten out. More formal techniques like Akaike Information Criterion [14] can be used to calculate optimal value of p. The advantages of the AR approach are (a) AR model can be used when train and test time series are of different lengths, (b) it does not require a distance measure used in similarity calculations in window and clustering based techniques, and (c) it captures the trend of time series using the AR coefficients.

We have chosen the above AR technique to detect anomalies using our framework. After a series of experiments with different orders for minimizing errors, which are not reported here for space limitations, we chose the AR model of order 5 to detect anomalies in our application metrics. For each workflow application we studied, we first made training runs of the workflows to collect the training time series data. The controlled testbed environment provided by ExoGENI was critical to obtaining "clean" training series data. The AR coefficients, and hence the AR(p) model parameters were calculated based on this offline training data. During future executions of the workflow applications, we used the AR model to predict the value for the next time instant and compared it with the actual online monitoring data for errors. When errors exceeded a certain threshold (say 20%), we generated anomaly triggers.

IV. Evaluation

In this section, we present an evaluation of the AR based anomaly detection approach for several example scientific workflows and applications running on the ExoGENI cloud testbed. For each example, we created artificial loads using competing applications or system tools while the workflow application is executing, and ran the AR based algorithm to determine errors from predicted metric values to pinpoint the time when an anomaly is triggered. All workflows and applications were pre-installed in virtual machine images. For the experiments, we used these images to instantiate HTCondor based virtual clusters on-demand on the ExoGENI rack at the Wayne State University (WSU), and then used Pegasus to plan and execute the workflows. We now present the results for each workflow/application example.

A. Periodogram Workflow

The Kepler satellite [15] produces a light curve recording the brightness of stars over time. Analyzing these light curves to identify the periodic signals that arise from transiting exoplanets requires the computation of periodograms that reveal periodicities in the time-series data along with estimates of their significance. Periodograms are computationally intensive, and the volume of data generated by Kepler requires high-performance processing. The periodogram workflow [16] parallelizes and distributes the computation of periodograms using several different algorithms and parameter sets over many thousands of light curves.

The periodogram application is computationally intensive, and hence we used a CPU intensive anomaly load generator, a competing application called the Conjugate Gradient (CG)

class C program from the suite of NAS parallel benchmarks (NPB) [17]. We instantiated the CG class C program at 270 seconds since the start of Periodogram application. The online monitoring data published in InfluxDB includes time series for several metrics for each application job in the workflow. In this case, we monitored the time-series for the *utime* metric for the Periodogram application, which is a measure of the time scheduled for the application process in user space. Figure 4 shows plot of *utime* collected over the duration of Periodogram execution with CG class C anomaly running in the background. We observe that the *utime* of Periodogram almost halves since the start of CG anomaly at 270 seconds and rises in value after the end of anomaly.

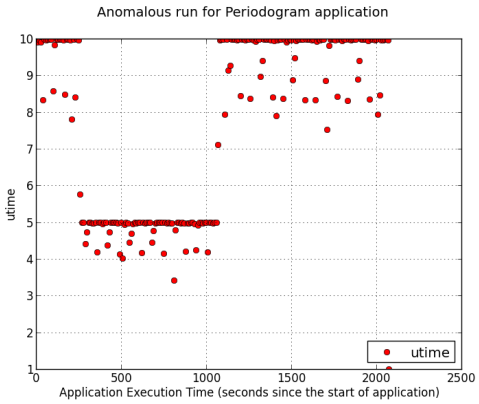

Fig. 4. Periodogram application with CG class C anomaly.

We calculated the AR coefficients using a training time series and generated the predicted time series. To detect the time at which anomaly was introduced, we plotted the error between predicted and the anomalous run at each second. Figure 5 shows the error plot in which X-axis represents the seconds since the start of Periodogram application and Y-axis represents the error of anomalous run from the predicted value. The error crosses the threshold of 20% at the 270 second mark. We can infer that our AR based anomaly detection technique is successfully able to detect the time at which anomaly was introduced.

B. Genomics Workflow

We used one of the analysis workflows used to process and prepare data generated by the UNC High-Throughput Sequencing Facility (HTSF) as the next example workflow. These workflows focus on a variety of applications, including full and partial genome assembly and alignment, variant detection, etc. We present results for two important applications in the genomic workflow.

1) Genomics gatk application: The gatk job in the genomics workflow stands for Tookit for Genome Anaylsis. The gatk job is CPU and memory intensive and hence we chose an anomaly load that stresses both the CPU and memory resources. We selected Multigrid (MG) class B program using a 256 x 256 x 256 grid from NPB suite as the anomaly load generator.

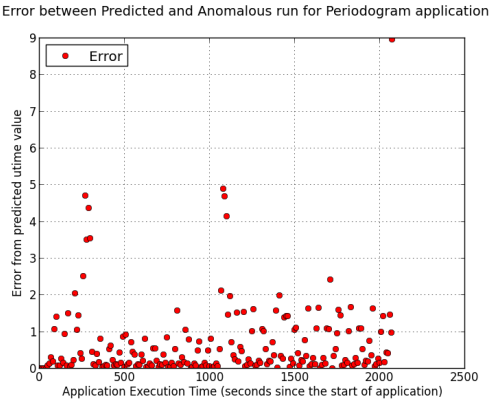

Fig. 5. Error plot for Periodogram application with CG class C anomaly.

For our anomalous run, we had 10 parallel threads each running MG class B program on the same HTCondor worker node which was running the gatk application. The anomalous load was introduced at 188 seconds into the execution of the job using shell scripts. As the gatk application is CPU and memory intensive, it starved due to resources being consumed by the anomalous load. To detect the anomaly, we collected the *utime* metric for gatk job which is shown in Figure 6. The effect of the anomalous load can be clearly seen in the *utime* drop in initial phase of the job run.

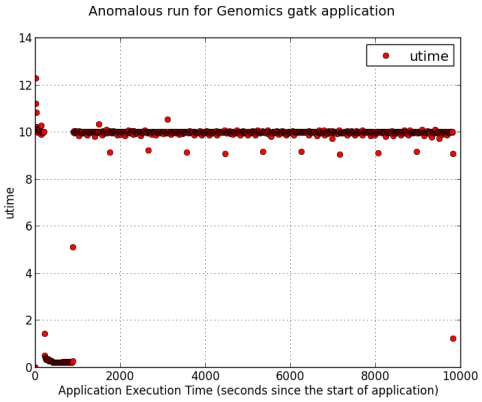

Fig. 6. Utime metric of gatk application with memory and CPU anomaly.

We calculated the AR coefficients from the training time series and generated the predicted time series. Figure 7 shows the error between the predicted time series and the anomalous time series. The error crosses the threshold at 220 seconds on the time axis. We had introduced the anomaly at 188 seconds and our approach detects it at 220 seconds into the run. The possible reason for that is although the scripts were initiated at 188 seconds, the MG program itself requires some initialization and startup time. This causes a small delay till the actual CPU and memory load kicks in. However, detecting the anomaly within 32 seconds after it was introduced is still quite useful considering the gatk application in the genomics workflow runs for approximately 42 mins (2520 seconds).

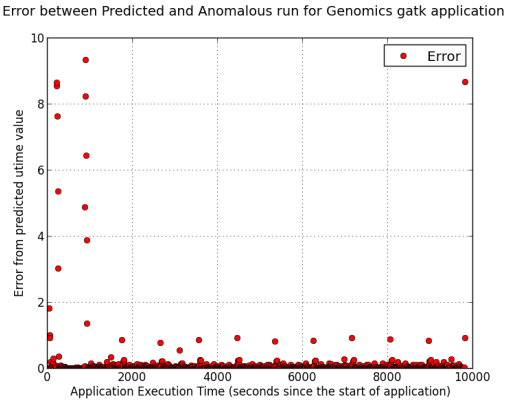

Fig. 7. Error plot of utime metric for gatk application with memory and CPU anomaly.

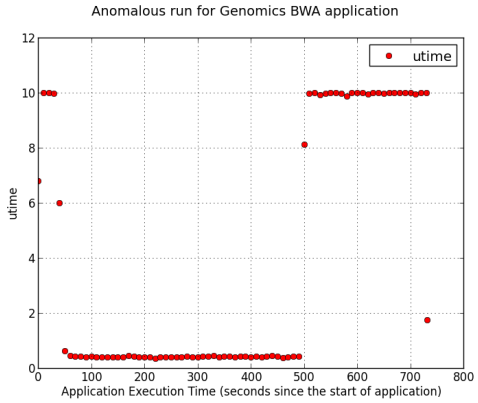

Fig. 8. Utime metric of BWA with memory and CPU anomaly.

2) Genomics BWA application: Burrows-Wheeler Aligner (BWA) is a program that aligns small genome sequences with larger reference sequences, such as human genome. The BWA application is a part of the genomics workflow. The BWA application is CPU and memory intensive. Similar to the gatk application, we used the MG class B program to stress the CPU and memory resources of the system to validate our AR based anomaly detection approach. BWA application requires higher percentage of CPU and memory resources during its run as compared to the gatk application. Hence we required only 5 threads of MG class B program running in parallel to create the bottleneck due to lack of enough CPU and memory resources. Each thread of MG class B program was instantiated 40 seconds since the start of BWA application and ran for a total duration of 7 minutes and 30 seconds ending at 490 seconds. To detect the anomaly, we collected the *utime* metric for BWA application over the entire duration of the application run.

Figure 8 shows the plot of *utime* over the execution of BWA application. The X-axis represents seconds since the start of BWA application and Y-axis represents the *utime* usage at each second by BWA application. We can clearly observe the strong effect of the introduced anomaly at 40 seconds on X-axis.

Similar to previous experiments, we calculated the AR coefficients from training time series and generated the predicted time series. In order to correctly determine when the anomaly was introduced we plotted the error from the predicted values at each second of the application run. The point where error threshold first crosses 20% is the time at which anomaly was introduced. Figure 9 shows the plot of error between predicted and anomalous values at each second of execution of BWA application. The error values first cross the threshold at the 50 second mark. We introduced the anomaly at 40 second since the start of BWA, and it was detected at 50 second. In Figure 9, we can also see some points cross the error threshold near the 500 second mark. These points indicate the end of introduced anomaly at the 500 second mark according to our detection technique. The anomaly was stopped at 490 seconds using shell scripts and our AR based approach detects end of anomaly at 500 seconds. This experiment shows that our AR based anomaly detection technique is not only able to detect the anomaly with good accuracy it is also able to determine the duration the anomaly was running. In some cases, knowing both the time of introduction and duration of anomaly can be highly useful to determine the cause of the anomaly.

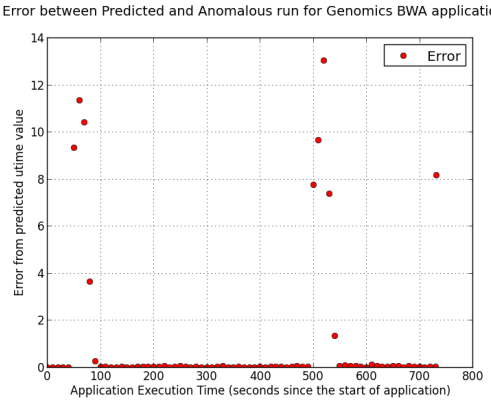

Fig. 9. Error plot of utime metric for BWA with memory and CPU anomaly.

C. Block Tri-diagonal Solver

We used the Block Tri-diagonal (BT) solver class C from the NPB suite as the next example. The BT program is CPU intensive, and we used the *stress* benchmark [18], which is a workload generator for POSIX systems, and can be used to create artificial CPU load. We configured *stress* to use 4 threads continuously running the sqrt() function to introduce anomaly 480 seconds into the BT program. We collected the *utime* metric for BT program using the /proc/pid/stat file in linux, where pid stands for the process ID. Figure 10 shows the graph of *utime* collected over the duration of BT program while running the *stress* anomaly.

We observe that there is sudden drop in *utime* after approximately 480 seconds. This sudden drop in *utime* is due to aforementioned CPU load introduced using the *stress* tool. As

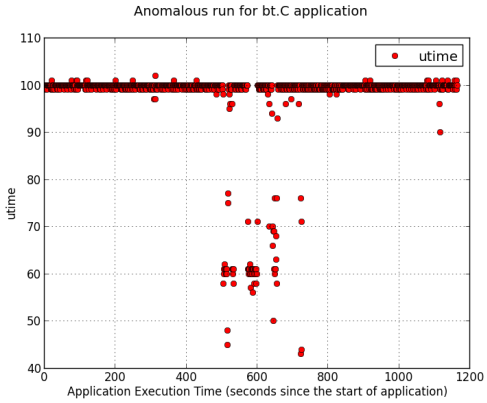

Fig. 10. BT class C program with CPU stress anomaly.

explained before, we generate a predicted time series based on AR coefficients from training time series. We also plot the difference of anomalous input time series and predicted time series as error at each time instant. Figure 11 shows the error plot of the *utime* metric. As seen in the figure, when error threshold crosses 20% at 480 seconds, we can conclude that anomaly was introduced at that point of time. Our approach is correctly able to pinpoint the precise time at which anomaly was introduced in the BT program.

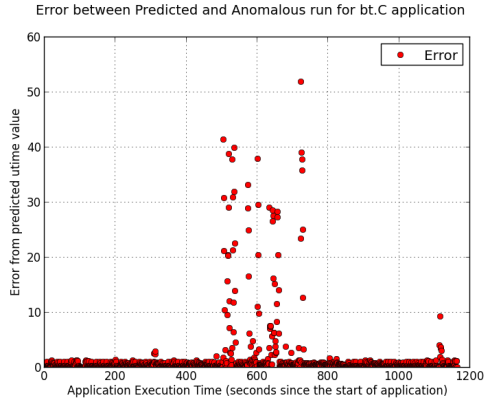

Fig. 11. Error plot for BT class C program with CPU stress anomaly.

We have also investigated AR based detection for anomalies during execution of I/O intensive benchmarks (FIO [19]) and applications (like NAMD molecular simulations), specifically for metrics like *write bytes* and *iowait*, a measure of time spent for waiting on I/O, when being injected with external I/O load using *stress*. The AR models can identify beginning and ending of anomaly for these cases. We are omitting these results for space constraints.

V. Related Work

Although different methods and techniques have been researched to detect anomalies, a majority of them are either machine learning or statistical techniques. Tan [20] discusses machine learning techniques - Tree Augmented Networks and Bayesian classifiers for predicting anomalies. Dean et. al. [21] present anomaly detection in IaaS clouds using unsupervised machine learning technique called Self Organizing Maps to predict emergent system behaviors.

Gaussian based statistical techniques use distribution mean μ and standard deviation of distribution σ to detect the outlier points [22], [23]. Students t-test [23] and Grubbs test [24], among other Gaussian based techniques, use the probability density function to calculate the estimated μ and σ for each observation, to detect if its anomalous.

Regression based statistical methods generate model parameters to fit the data and calculate error based on predicted and actual value. Our AR based anomaly detection technique falls under the category of regression based statistical techniques. Bianco et. al. [25] and Chen et. al. [26] discuss anomaly detection using another variant of regression model called the Autoregressive Integrated Moving Average (ARIMA) model. The AR method in conjunction with other methods has been previously used to detect anomalies. One such example is [27], where the authors discuss about building probabilistic models using relevance vector and Auto Regression (AR) to detect anomalies. They use the probability density of the data points to detect anomalies. To our knowledge, the work discussed in this paper is the only work that describes the application of AR based anomaly detection techniques on scientific workflows to pinpoint the time at which anomaly was introduced with marginal error.

Extensive research has been done in qualitative performance evaluation of scientific workflow applications. Buneci [28] discusses the performance evaluation approaches to assess if overall application behavior is as expected. This work describes methods to extract the temporal information like patterns and variance of performance time series to predict behaviors in long running scientific applications.

Although, our paper focusses mainly on anomaly detection, there has been considerable work done on fault handling and fault tolerant systems for scientific workflows. Bala and Chana [29] present use of failure prediction models to predict task failures while running scientific workflows on cloud services. Their work evaluates different machine learning models and suggests the use of Naive Bayes model over other models for good accuracy of failure predictions. Sometimes, the job failures occur at the workflow level itself, and Samak et. al. [30] propose the use of k-means clustering algorithm to identify workflow classes with high failure percentages. The online prediction system developed in [30] classifies if the running workflow is high failure percentage class or not. Crawl and Altintas [31] propose methods to record data dependencies for provenance information in Kepler scientific workflow system, and describe how it can be used for failure recovery. Chen and Deelman [32] propose a task failure modeling framework that uses Maximum Likelihood estimation method to model workflow performance. They present three different fault tolerant task clustering strategies to improve the performance of scientific workflows.

VI. Conclusions and Future Work

In this work, we presented a system for detecting anomalies when scientific workflows and applications execute on networked clouds. We described our monitoring data collection framework that uses new Pegasus monitoring capabilities to obtain time series data for various performance metrics relevant to the workflow application performance. We presented our anomaly detection algorithm based on auto-regression technique, which can analyze the time-series data and generate anomaly triggers when errors from predicted AR model in the online monitoring data exceed a certain threshold. We presented an evaluation of the AR based detection technique using example workflows from genomics, astronomy and NAS parallel benchmarks, and showed that our approach can effectively pinpoint anomalies when external load is introduced. In future, we plan to extend the work to detect other kinds of anomalies, for eg. workflow level anomalies, data transfer anomalies, which are also observed in real systems. We also plan to investigate other kinds of detection algorithms. We plan to leverage this work to pinpoint the sources of anomalies, and develop different kinds of adaptation mechanisms to respond to observed anomalies.

Acknowledgments

Work for this paper was supported by several grants - DoE ASCR Panorama (DE-SC0012390), DoE SciDAC SUPER (DE-FG02-11ER26050/DE-SC0006925), and the NSF GENI (#1872).

References

[1] Amazon Elastic Compute Cloud (Amazon EC2), http://www.amazon.com/ec2.

[2] OpenStack Cloud Software, http://openstack.org.

[3] I. Baldine, Y. Xin, A. Mandal, P. Ruth, A. Yumerefendi, and J. Chase, "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 (TRIDENTCOM)*, 2012.

[4] "PANORAMA: predictive modeling and diagnostic monitoring of extreme science workflows," http://sites.google.com/site/panoramaofworkflows.

[5] J. Chase, L.Grit, D.Irwin, V.Marupadi, P.Shivam, and A.Yumerefendi, "Beyond virtual data centers: Toward an open resource control architecture," in *Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library)*, May 2007.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[7] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh *et al.*, "Mapping abstract complex workflows onto grid environments," *Journal of Grid Computing*, vol. 1, no. 1, pp. 25–39, 2003.

[8] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[9] Exogeni Wiki, https://wiki.exogeni.net/.

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

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

[12] 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.

[13] "InfluxDB," https://influxdata.com/.

[14] C. M. Hurvich and C.-L. Tsai, "Regression and time series model selection in small samples," *Biometrika*, vol. 76, no. 2, pp. 297–307, 1989. [Online]. Available: http://biomet.oxfordjournals.org/content/76/2/297.abstract

[15] "Kepler: A Search for Habitable Planets," http://kepler.nasa.gov.

[16] G. B. Berriman, G. Juve, E. Deelman, M. Regelson, and P. Plavchan, "The application of cloud computing to astronomy: A study of cost and performance," in *In Workshop on e-Science Challenges in Astronomy and Astrophysics*, 2010.

[17] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks—summary and preliminary results," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '91. New York, NY, USA: ACM, 1991, pp. 158–165. [Online]. Available: http://doi.acm.org/10.1145/125826.125925

[18] "stress workload generator," http://people.seas.harvard.edu/ apw/stress/.

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

[20] Y. Tan, "Online performance anomaly prediction and prevention for complex distributed systems," Ph.D. dissertation, 2012, aAI3538490.

[21] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th International Conference on Autonomic Computing*, ser. ICAC '12. New York, NY, USA: ACM, 2012, pp. 191–200. [Online]. Available: http://doi.acm.org/10.1145/2371536.2371572

[22] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[23] M. Markou and S. Singh, "Novelty detection: A review—part 1: Statistical approaches," *Signal Process.*, vol. 83, no. 12, pp. 2481–2497, Dec. 2003. [Online]. Available: http://dx.doi.org/10.1016/j.sigpro.2003.07.018

[24] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, pp. 1–21, 1969.

[25] A. M. Bianco, M. Garca Ben, E. J. Martnez, and V. J. Yohai, "Outlier detection in regression models with arima errors using robust estimates," *Journal of Forecasting*, vol. 20, no. 8, pp. 565–579, 2001. [Online]. Available: http://dx.doi.org/10.1002/for.768

[26] D. Chen, X. Shao, B. Hu, and Q. Su, "Simultaneous wavelength selection and outlier detection in multivariate regression of near-infrared spectra," *Analytical sciences : the international journal of the Japan Society for Analytical Chemistry*, vol. 21, no. 2, pp. 161–166, February 2005. [Online]. Available: http://dx.doi.org/10.2116/analsci.21.161

[27] R. Fujimaki, T. Yairi, and K. Machida, *Advances in Knowledge Discovery and Data Mining: 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ch. An Anomaly Detection Method for Spacecraft Using Relevance Vector Learning, pp. 785–790. [Online]. Available: http://dx.doi.org/10.1007/11430919_92

[28] E. S. Buneci, "Qualitative performance analysis for large-scale scientific workflows," Ph.D. dissertation, Duke University, 2008.

[29] A. Bala and I. Chana, "Intelligent failure prediction models for scientific workflows," *Expert Syst. Appl.*, vol. 42, no. 3, pp. 980–989, Feb. 2015.

[30] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, G. Mehta, F. Silva, and K. Vahi, "Online fault and anomaly detection for large-scale scientific workflows," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, Sept 2011, pp. 373–381.

[31] D. Crawl and I. Altintas, *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers*, 2008, ch. A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows, pp. 152–159.

[32] W. Chen and E. Deelman, "Fault tolerant clustering in scientific workflows," in *Services (SERVICES), 2012 IEEE Eighth World Congress on*, June 2012, pp. 9–16.