Failure Analysis of Distributed Scientific Workflows Executing in the Cloud

Taghrid Samak*, Dan Gunter*, Monte Goode*, Ewa Deelman‡, Gideon Juve†,
Fabio Silva‡, Karan Vahi‡
*Lawrence Berkeley National Laboratory, Berkeley, CA, USA
†University of Southern California, Marina Del Rey, CA, USA
‡University of Southern California Information Science Institute, Marina Del Rey, CA, USA

***Abstract*—This work presents models characterizing failures observed during the execution of large scientific applications on Amazon EC2. Scientific workflows are used as the underlying abstraction for application representations. As scientific workflows scale to hundreds of thousands of distinct tasks, failures due to software and hardware faults become increasingly common.**

We study job failure models for data collected from 4 scientific applications, by our Stampede framework. In particular, we show that a Naive Bayes classifier can accurately predict the failure probability of jobs. The models allow us to predict job failures for a given execution resource and then use these failure predictions for two higher-level goals: (1) to suggest a better job assignment, and (2) to provide quantitative feedback to the workflow component developer about the robustness of their application codes.

I. Introduction

Scientific applications today make use of distributed resources such as campus resources, grids, clouds, or a mixture of them all. The size and complexity of resources and computations needed to support these applications continues to grow. This complexity brings increased potential for failures and performance problems. Virtualized resource management, or "cloud", technologies can simplify the application's view of the resources, but do not remove the complexity inherent in large-scale applications.

In large-scale workflows, the tools available in current workflow engines for interactively looking at which nodes failed are not sufficient to cope with serious failures, because by the time a significant pattern of failure is detected large amounts of resources (often including scientists' time waiting for workflows to complete) will have been wasted. The NSF Synthesized Tools for Archiving, Monitoring Performance and Enhanced DEbugging (Stampede) project addresses this gap by providing online and automated failure analysis. In our previous work [12], we extracted monitoring data from all the major components – hardware, software, and OS – organized them efficiently, and then augmented them with novel work in online, automated failure analysis.

To this point our failure analysis has used explicit failure signals from workflow nodes. The next logical step is to understand application failures and performance with respect to interactions between components. To perform this task at scale, one needs models that automate the discovery of the most important variables and interactions. In this paper we examine the execution logs from a number of applications that ran on instances of the Amazon EC2 [1] cloud infrastructure. These applications were managed by the Pegasus Workflow Management System [10].

Virtualized execution environments, including but not limited to "clouds", are of particular interest today due to their growing popularity at all scales of scientific computing. In virtualized environments many levels of failures are hidden from the application, and thus hard to monitor and analyze. Detailed system statistics can be difficult or impossible to obtain. Thus, a system that could provide approximations of failures without detailed system statistics, i.e. using only application-level metrics, would be a desirable tool in cloud environments.

In clouds, multiple VMs may be associated with a single physical host. With VM migration, a single VM may even be associated with multiple different hosts during its lifetime. In the analysis presented here, the distinction between host and VM is abstracted. To avoid confusion, we henceforth use the term *Execution Resource* (ER) to mean an application VM running on, and maybe sharing, a physical host.

The results presented here extend our previous work [22], [23], where we addressed the problem of detecting failing workflows top-down, using high level aggregation of run-time statistics such as whether jobs have failed or succeeded. After a workflow has been identified as failing with high statistical significance, we used this information to perform detailed root-cause analysis at the job level. By focusing on a failing workflow, we were able to use the history of jobs executed by this workflow to learn an overall job behavior and predict job failures.

Contribution: This work presents a bottom-up approach to job failure analysis in virtualized environments, where aggregated statistics from individual ERs are used to predict job failures. We first present models for job failures on ERs as a decentralized analysis approach that can be deployed on individual ERs and can be executed while the workflow is running. We then present our analysis of job failure models using combined information from multiple ERs.

Three main outcomes are sought from both analysis levels:

978-3-901882-48-7

1) predicting job failures given a certain ER assignment by the execution engine, 2) using failure predictions to suggest a better job assignment, and 3) providing feedback to the workflow component developer about the robustness of their application codes.

The rest of this paper is organized as follows. The data presented in this paper are collected using our Stampede framework [12], which is described in Section II along with the workflow data model. We take into consideration four different types of workflows executed a number of times. The analysis approach along with experiment description is presented in Section III. The results are presented in Sections IV and V. We survey related work in Section VI, and conclude in Section VII.

II. Background

The main focus of this work is the modeling of scientific application failures as they may occur in cloud environments like Amazon EC2. In our previous work, we have implemented the necessary infrastructure to monitor, in real-time, large workflows running on grids, clusters and clouds. The infrastructure relies on the Pegasus [10] workflow management system, and the NetLogger [13] monitoring toolkit. The system is described in the next subsection, followed by the workflow data model used in the analysis.

The performance of Pegasus in running scientific workflows on Amazon EC2 has been studied in [17], [26]. The main focus was on resource selection to optimize the running time of the workflow. We extend this analysis by investigating failure models in the same cloud environment.

A. Stampede Analysis System

The Stampede analysis system is shown in Figure 1. These components include the workflow execution engine, log collection, and analysis. The workflow engine and execution components are part of Pegasus, described in detail elsewhere [10]. The log collection and analysis components are described below. For more information, see [12], [22].

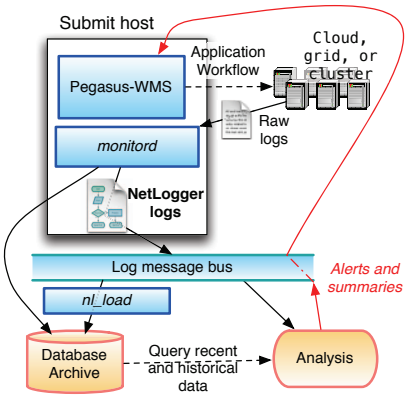


Fig. 1: Overview of the Stampede Architecture.

The analysis system was designed to be:

- Scalable: The system has been tested on large workflows with millions of jobs.
- Streaming: Data is processed as a stream of "events", which are a stream of semi-structured log messages in the NetLogger Best Practices (BP) [6] format by a program called *pegasus-monitord*.
- General-purpose: The part of the architecture that is specific to Pegasus is contained in the *Submit host* box in Figure 1. The rest of the architecture, starting from the log message bus, is abstracted from the execution engine. Resource-specific data, e.g. from Kickstart [27], are renamed into generic forms.
- Queryable: Data is stored in a relational archive, which normalizes the log events into a close approximation of the attributes and relationships of the data model described in Section II-B.

The analysis components (bottom right of Figure 1) can obtain data in two ways: they can query the database or subscribe to a stream of log messages from the message bus. We have verified that for either data path the performance of the analysis tools is sufficient for online analysis [22]. This is important because the goal of this work is to enable dynamic analysis and adjustment of running workflows. To perform our statistical analyses, we primarily use the *R* [4] analysis environment, which has rich and and well-tested libraries of statistical and learning functions.

B. Workflow Data Model

We have developed a data model of workflows for use across all analyses and storage representations. Here we present only relevant aspects; a more complete terminology is described in [12]. In this model, each workflow is represented by two connected sub-models: an *abstract workflow* and an *executable workflow*. The *abstract workflow* describes the computations, data dependencies in a resource-independent way.

An *executable workflow* is an instantiation of a workflow for a particular set of target execution resources. In the executable workflow, data movement operations are added, the logical tasks are mapped to concrete resources, executables are specified, etc. This mapping can be complex: logical tasks may be joined together or optimized away entirely. New concrete tasks may be added to create directories and manage data staging and registration. A *job* is a concrete task within an executable workflow that initializes or cleans up the application, or runs application code (one or more programs). The *job type* indicates either the general function of the job (e.g., "cleanup") or, in the case of application jobs, the type of program (e.g., "rspectra"). In this analysis, we focus primarily on jobs and executable workflows, henceforth simply "workflows".

III. Analysis Approach

This section provides an overview of the analysis approach, the data sets used and the evaluation methodology. Subsequent sections will describe the application of this methodology.

A. ER-Based Failure Analysis

Our analysis approach is focused on detecting the failures of *jobs*, as defined in Section II-B, and relating those failures to the *job type* and ERs for a given workflow. The output of the analysis is a model for job failure behavior. This model provides (a) guidelines for the execution engine to make better scheduling decisions, and (b) a basis for high-level summaries for the application user.

We formalize the problem as a general classification problem: $\mathcal{C} = \langle \mathcal{S}, \mathcal{T}, \mathcal{H}, \rho \rangle$ where

- $\mathcal{S} = \{\texttt{SUCCESS}, \texttt{FAILURE}\}$ is the set of possible job completion statuses.
- $\mathcal{T} = \{t_1, t_2, \ldots t_k\}$ is the set of job types as defined by the application.
- $\mathcal{H} = \{h_1, h_2, \ldots h_l\}$ is the set of available ERs.
- $\rho : \mathcal{T} \times \mathcal{H} \mapsto \mathcal{S}$ is a classification function, mapping tuples of $< t_i, h_j >$ to completion status.

A learning phase will use information from previous job executions to build a model for ρ. We focus on job type as an application-level feature, and evaluate the prediction capability of the completion status on different ERs. Other job-specific information can be added to the problem definition, which will result in a more complex learning phase and models. We aim here to investigate the simple two-feature models and evaluate their performance.

We chose Naive Bayes classifier as the underlying inference model for ER information. This classifier assumes independence for each marginal probabilities, which is justified for job information (jobs are application-level abstractions). The classifier can also easily scale for more parameters. We first describe the general model, and then explain the process used in analyzing the data collected for scientific workflows running on EC2.

The learning process aims to build a model for the posterior probabilities of the classifier. In our problem domain, we wish to predict job completion status given job attributes and execution environment. We found that using only job type, and ER identifiers was sufficient to accurately predict job status, and also improve job placement strategy.

Using Bayes' theorem, we can compute the probability of `SUCCESS|FAILURE` of a certain job type, when executing on a certain ER by:

$$p(S|T,H) = \frac{p(S)p(T,H|S)}{p(T,H)} \quad (1)$$

The model is rebuilt in real-time by using previous job information to predict whether an incoming job will succeed or fail. The probabilities are estimated after each job completion (success or failure). In other words, each completed job is added to the learning set, and the model is recomputed.

For a large number of ERs, this can become a costly process. Thus, it is more efficient to compute models of job failures for individual ERs, without complete knowledge of the entire workflow execution. Each ER constructs a job failure model for each job type, considering only its local view of previous job executions.

At each individual ER, the posterior probabilities of failures can be computed given each job type as:

$$p(S|T) = \frac{p(S)p(T|S)}{p(T)} \quad (2)$$

Parameter estimation for single or multiple ERs (Equations 2 and 1 respectively) was performed using Laplace smoothing in the learning phase, to avoid zero probabilities.

B. Experimental Data

Stampede is currently used in the analyses of many scientific applications. The analysis presented here has been performed on 157 real workflow executions on Amazon EC2, across four distinct applications: Broadband, Epigenome, Montage, and Periodograms.

Amazon EC2 offers several resource configurations for virtual machine instances. Each instance type is configured with a specific amount of memory, CPU, and local storage. For the dataset presented here, the c1.xlarge instance type is used. This type is equipped with two quad core 2.33-2.66 GHz Xeon processors (8 cores total), 7 GB RAM, and 1690 GB local disk storage. For more details on the experiments and EC2 performance, refer to [17]. For this paper, we focus on the failure aspects of each application.

The datasets collected with Stampede for the described experiments are summarized in Table I. A brief description of each application follows.

TABLE I: Summary of datasets for each application

	Cumulative Counts		
Application	**Workflows**	**Jobs**	**Tasks**
Broadband	51	25,305	70,628
Epigenome	16	4,726	19,469
Montage	45	9,088	520,808
Periodograms	45	76,147	1,894,876
TOTAL	157	115,266	2,505,781

Broadband [2] is a computational platform used by the Southern California Earthquake Center to simulate the impact of an earthquake at one of the Southern California faultlines, to guide the building design procedures in a given area. The workflow has four stages of computation: rupture generation, low-frequency seismograms, high-frequency seismograms, and finally a stage that extracts parameters from the seismograms that are of interest to earthquake engineers. The number of executing nodes roughly triples from the first to third stage, then each seismogram is extracted in parallel. All stages have some computation and I/O, but the seismogram steps are the most demanding and the high-frequency seismograms consume roughly half the total execution time.

Epigenome workflows perform various genome sequencing operations using the DNA sequence data generated by the

Illumina-Solexa Genetic Analyzer system [5]. These workflows have an initial data splitting stage, then a data-parallel pipeline, and then a merge and serial processing of the results. Most of the runtime is spent in the final parallel stage, *map*, that aligns sequences with the reference genome; but the final serial stages are also non-trivial, taking together about a quarter of the total runtime.

Montage was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to generate custom mosaics of the sky [7]. Montage workflows re-project, rotate, and normalize levels in the input images to form the output mosaic. The workflow structure for Montage has two expansion-contraction phases: in the first phase, overlapping input images are subtracted, typically involving more parallel jobs than images, and the results of these subtractions are merged. In the second phase, the merged images are used to calculate a background image that is subtracted (in parallel) from each, and these results are merged into the final mosaic of the sky. The background calculation and final mosaic calculation take the majority of the runtime, as they are computationally intensive and only partially parallel.

Periodograms calculate the significance of different frequencies in time-series data (light curves) to identify periodic signals [3]. The periodigram application designed by IPAC (Caltech) is being used today to search for exoplanets in the Kepler mission's data. The processing of each frequency sampled in a periodogram is performed independently of all other frequencies, and so periodogram calculations are performed by highly parallel workflows. However, the light curves may have widely varying number of data points, so the parallel jobs may have widely varying runtimes.

C. Evaluation Methodology

Pegasus makes scheduling decision for each job, and maps its execution to one of the available ERs. We evaluated both individual ER analysis and multiple ERs on the aforementioned wide range of applications. The following steps summarize the evaluation process.

We used the following data for each job: job submit time, job type, final job status (success or failure), and the ER on which the job was actually executed. The model provides two main marginal probabilities for `JOB_SUCCESS` and `JOB_FAILURE`.

For a particular workflow, and at any point in time, we use jobs that have previously completed (successfully or failed) to build a Naive Bayes model for job status, to predict incoming job behavior. The models are built either on individual ERs (basic model), or by combining information for multiple ERs (combined model). When a new job is submitted at a subsequent scheduling step, predictions are made considering two main aspects: job type, and ER information. Both attributes are used to predict job status. We evaluate the models for the same job against all available ERs, to choose which ER will maximize the likelihood of success. Since we evaluate our model on an off-line dataset, we are able to evaluate the accuracy of the prediction in two ways:

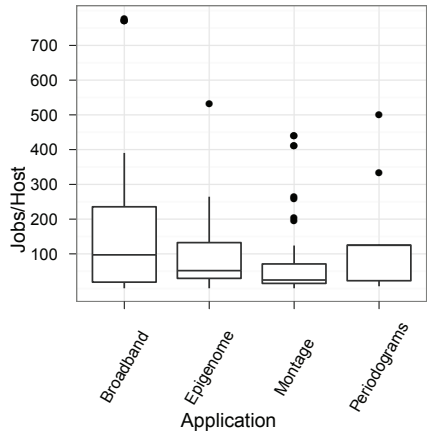


Fig. 2: Distribution of Jobs per ER for each applications.

- **Prediction performance:** Comparing the predicted status with the actual job completion status, which corresponds to evaluating the prediction accuracy.
- **Improving Pegasus:** Comparing predicted status on *any* ER with the actual job status. If the prediction is accurate, this can help the workflow execution system pick the best ER, i.e. the one with the lowest probability of failure, for a given job.

IV. Single Execution Resource (ER) Evaluation

This section presents the results of applying a Naive Bayes classifier to ER-based measurements. The classifier takes as input the job types, $\mathcal{T}$, and ERs, $\mathcal{H}$, and builds a model for a classification function, ρ, that maps these inputs to a completion status $\mathcal{S}$, thus: $\rho := \mathcal{T} \times \mathcal{H} \mapsto \mathcal{S}$.

The main goal is to characterize job failure behavior for each individual ER. A model of job state (failed or successful) is built for each ER, based on job type. This model is then used for each incoming job to predict whether it will succeed.

We first present some exploratory analysis for the available workflow executions. Figure 2 summarizes the total number of jobs executed per ER for each application. In this dataset, *Periodograms* had the most number of job instances, and also used the largest number of ERs. *Broadband* workflows had the maximum throughput (maximum number of jobs per ER).

As our analysis aims to characterize failures, we will focus on ERs with frequent job failures. The rate of ERs which had at least a single job failure were similar for *Broadband* (39%), *Epigenome* (30%), and *Montage* (31%). *Periodograms* had the lowest percentage of ER failures at 0.3%; in fact this was only a single ER, with 2 job failures. In the coming sections, *Periodograms* will not be considered for further analysis.

Our metrics are as follows. The *accuracy* is measured for both successful and failed jobs, and computed as the percentage of correct classification from all jobs. Errors are measured by the standard false positive and false negative rates. Here, we consider the `FAILURE` as the reference class.

Application	Accuracy	FP	FN
Broadband	0.73	0.27	0
Epigenome	0.81	0.19	0
Montage	0.69	0.31	0

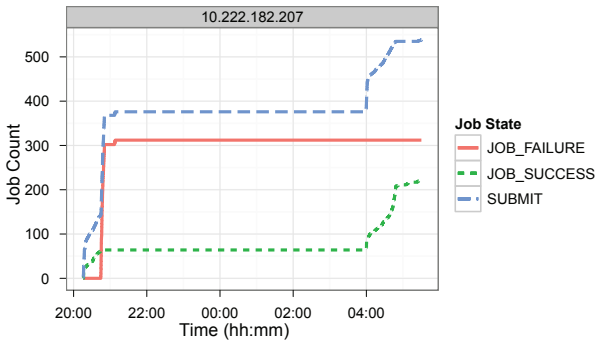
TABLE II: Prediction accuracy for job failures using individual ER inference. FP and FN represent false positive and false negative rates, respectively.

Thus, *false positives* (FP) are jobs classified as `FAILURE` that actually finished successfully and *false negatives* (FN) are jobs classified as `SUCCESS` that failed. Both FP and FN are computed as ratios by dividing by the total number of jobs.

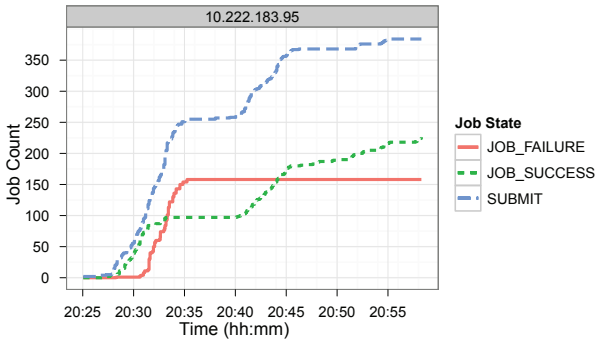
Table II shows the prediction accuracy, FP, and FN for each application. The learning phase models job failures at each ER, based solely on past job executions on the same ER. The accuracy was over 65% for all applications. *Epigenome* achieved the highest accuracy, and lowest FP. Higher values of FP over FN indicate that the model will take a safe decision by predicting a job as failed more often than successful. This is not desirable on large scale, as it means that successful jobs might be classified as failed, which results in wasted resources. On the other hand, the low FN indicates that a failed job will rarely go undetected. Notice that the experiments here were performed on a small number of ERs for each run. With a much larger pool of resources, the FN rate might increase.

In addition to job classification, we further investigate ERs with maximum number of job failures. Figure 3 shows the cumulative number of jobs executed at each ER over time. For *Broadband*, the number of job failures increased at the start of the ER operation then remained constant at 300. The success rate then increased. As we will see in the following section, the trend is similar to some job failure probability trends. Unfortunately, it is only visible when looking at the overall ER behavior, not specific job types. Similar observations hold for *Epigenome*.

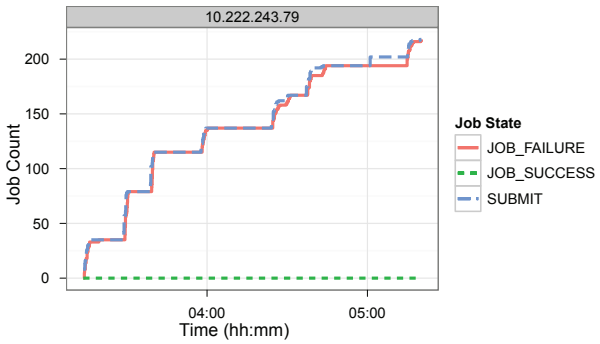
On the other hand, the number of failures for the *Montage* ER is increasing monotonically, with no successful job completions. Predicting job failures independently for this ER would suggest that there is a major problem with every job type submitted to this ER. In this case, the ER itself might be the cause of the problem. By performing multiple ER analysis, the problem can be identified, and jobs can be scheduled on other ERs. *Montage* workflows will be discussed in more detail in the following section.



(a) Broadband



(b) Epigenome



(c) Montage

Fig. 3: The number of jobs submitted, succeeded and failed on the ERs with maximum failures for each application.

V. Combined Model Evaluation

In this section, we evaluate the performance of the combined model, where multiple ERs are used in building failure models. First we evaluate the prediction accuracy, then proceed to predict better job placement strategies.

A. Prediction Performance

Our first measure of the combined model is the prediction performance. When a job is assigned to be executed on a specific ER, we predict the final completion status using the model learned from all previous jobs, considering all ERs.

Recall that the evaluation is performed on the off-line data from our experiments. We apply the model incrementally, but evaluate the final accuracy, FP, and FN on the complete dataset for each application. Table III shows the average performance measures for *Broadband*, *Epigenome*, and *Montage*. The failure prediction model for *Epigenome* was the most accurate, without any FPs. This means no jobs were predicted to fail that did succeed, which is important in practice so that jobs will not

be unnecessarily rescheduled. *Epigenome* had also the smallest FN rate. Predictions for *Broadband* were the least accurate, with high error rates. This was expected, as the number of jobs per individual ER was much larger than other applications (see Figure 2).

Comparing those results with single ER evaluation shows that the FP rate has dropped for all applications, but the FN increased. It is important to note that the drop in FP is very beneficial, as it means less successful jobs are canceled (mistakenly). Missing more failed jobs, i.e. higher FN, reduces efficiency, but any FN rate below 1 is still an improvement over not running the analysis at all.

Application	Accuracy	FP	FN
Broadband	0.74	0.10	0.16
Epigenome	0.96	0.00	0.04
Montage	0.84	0.04	0.12

TABLE III: Prediction accuracy for job failures using the combined model. FP and FN represent false positive and false negative rates, respectively.

B. Dynamic Execution Resource (ER) Selection

In this section, we look at the details of individual workflow executions, and analyze job behavior over time. For each application, we show results for the probability of job failure over time for the combined model, considering all available ERs. These results have two major benefits for the execution:

- The probability of failures estimated for each job type, for different ERs, can guide the workflow's job placement.
- The failure behaviors of different job types can be used to diagnose application-level problems by correlating failures directly with high-level workflow components.

We also compare the results from our previous work on high-level workflow analysis [22], [23], which aimed to predict the overall workflow failure, then performed root-cause determination. The examples shown here demonstrate that low-level analysis can sometimes help identify problems earlier.

1) Broadband: Figure 4 shows an example workflow from *Broadband*. Each panel shows failure probabilities for a specific job type. The x-axis corresponds to the life time of the running workflow, where the model is evaluated for each job type, at each scheduling step. For each job type, the failure probability is computed for the ER assigned by Pegasus to that job (the black solid line). The predicted failure probabilities for all other ERs are estimated, and the minimum is reported (the red dashed line). The figure shows the potential improvement in job failure probability had the inference been done to test which ER would minimize failure probability.

It is important to note here that the prediction pronounces the job as doomed to failure when the probability is greater than 0.5 ($p(\texttt{FAILURE}) > p(\texttt{SUCCESS})$). By inspecting the failure probability over time, our target case would be when the prediction (red dashed line) falls often below 0.5 and

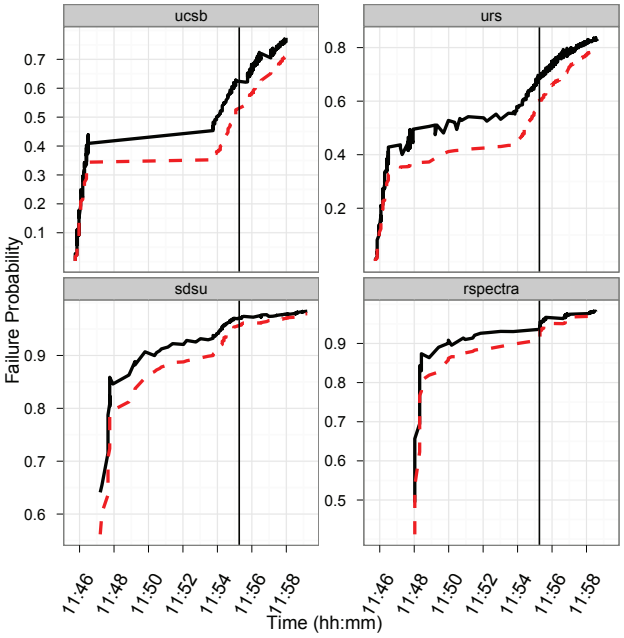


Fig. 4: *Broadband* id=43, multiple ERs inference.

the original probability (failure probability at the ER actually selected by Pegasus) is greater than 0.5.

The vertical line shows the detection time from our previous analysis [22], [23], where later inspection identified job type as one of the main factors deciding workflow failures. As clearly visible, the failure probability estimation on individual job types was able to identify the problem much earlier than high-level workflow aggregations, especially with the most problematic jobs (`sdsu` and `rspectra`).

By taking a closer look at each job type, we can potentially identify the root-cause of the problem. For example, `ucsb` and `urs` are facing increasing failure probability that stabilizes over time, then increases again. Moving the job to a different ER following the model recommendation (red dashed line) might save the job from later increased failures. The plateau effect for `ucsb` reflects a time interval where this job is not seen by the scheduler. This indicates dependencies on other jobs, or data movement. At the end of this plateau, (time $11:54$), an incoming job should be routed to an ER with lowest failure probability (dashed line). The stability of `urs` between time $11:46$ and $11:54$ has some variability, but the dashed line is stable, which provides earlier recommendation for changing the ER.

This does not apply to `sdsu` and `rspectra`, where the failure probabilities are high for earlier jobs (time $11:46$). Even with the intermediate stable trend, the failure probability is always > 0.5, no matter which resource is suggested. This indicates that the problem is with the workflow components and needs to be fixed at the application level.

Similar results are shown for another workflow run in Figure 5. Moving `ucsb` and `urs` to a different ER can save the

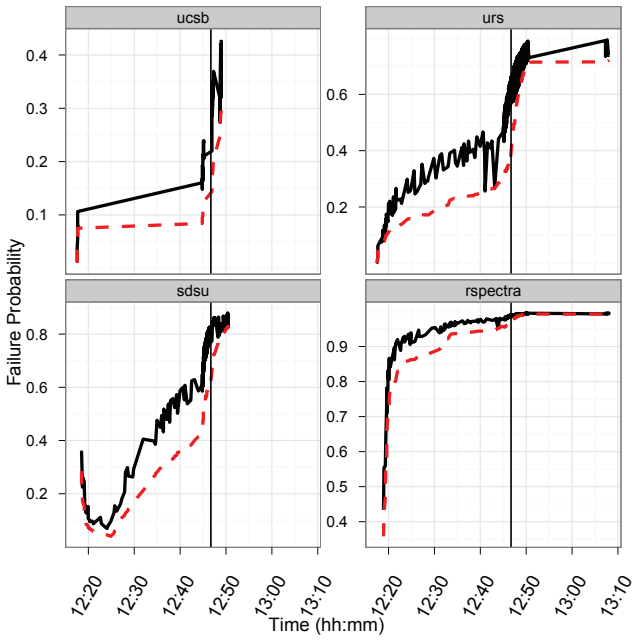


Fig. 5: *Broadband* id=44, multiple ERs inference.

job, which is indicated by the lower improved probability at the beginning. On the other hand, `sdsu` and `rspectra` need more application-level attention from the beginning, where the failure probability is high.

Comparing detection time with previous work (the vertical line), shows significant improvement, where moving jobs to different execution resource might save the entire workflow.

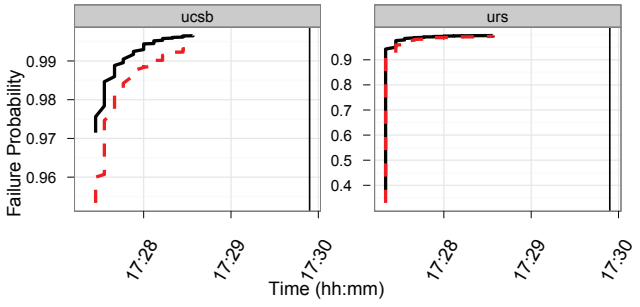


Fig. 6: *Broadband* id=29, multiple ERs inference.

Similar results were also obtained for a much smaller *Broadband* workflow (Figure 6), where the resource-based analysis identified failures much earlier than workflow-level prediction.

2) Epigenome: Figure 7 shows the job failure probability for one of *Epigenome* workflow executions (workflow 21). The general trend shows that the failure probability could be decreased by changing the ER, but not significantly. In most job types (`filterContams`, `sol2sanger`, `fast2bfq`), the failures were not significant to start with (less than 0.1 probability). Setting the 0.5 threshold will help Pegasus trigger ER assignment based on the model recommendation only when the failures have significant effect.

The same conclusion does not hold for the `map` job, where the failure probability is consistently increasing, which means that this particular job (or workflow component) needs more application-level analysis to be fixed. Its performance cannot be fixed by moving it to another ER.

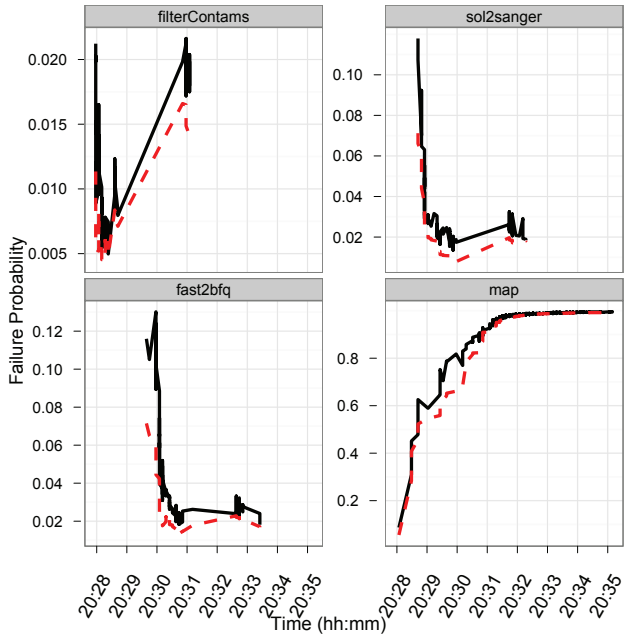


Fig. 7: *Epigenome* id=21, multiple ERs inference.

This *Epigenome* workflow run (21) is of particular interest, since the workflow has failed but the high-level prediction did not detect it in our previous work. The new analysis presented here shows that it is possible to still identify the problem and save the workflow.

3) Montage: The results from *Montage* workflows were different from *Broadband* and *Epigenome*. Most job types performed well (low failure probability), with the exception of `clean` jobs, which are added by Pegasus to delete files no longer needed at the execution site. Performance of the `clean` jobs is shown in Figure 8. The results from our analysis were later confirmed by our development logs for Pegasus, which show that for this *Montage* workflow, we were testing a new version of the `clean` job. Failures observed in the analysis reflect the bugs in a new version of the `clean` job, that were discovered and fixed in subsequent versions.

VI. Related Work

Failure analysis in grid environments have been studied extensively [8], [9], [14], [20], [28]. The work in [14] focused on anomaly detection in network operations, and disk IO. Detection of machine misconfiguration was the main focus in [20], where the output of simultaneously running jobs

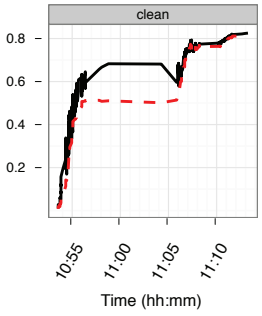


Fig. 8: `clean` jobs, *Montage* id=46, multiple ERs inference.

reflects machine conditions. Our approach focuses on job failures at particular machines. A bottom-up approach for grid workflow analysis was presented in [8]. Time series analysis of feature vectors from system measurements is applied to identify signatures for each job. A domain expert classifies those signatures, as expected or unexpected task behavior. Although our approach does not use job details in the analysis, the failure characteristics provide insight on high-level job abstractions, without the need for expert knowledge. Time series were also used in [28], where applications anomalies are identified. In [9], offline analysis was applied to detect root causes for job failures. Decision trees are used for postmortem identification of problem causes. The work can be contrasted to our previous work on job fault diagnosis [23], where online detection is applied. Our work here focuses on execution resources, and how to optimize resources selection based on failure patterns.

In high performance computing, system problems leading to job failure events were investigated in [18], [19], [24], [30]. The work in [19] focused on system failure events, and used machine learning to predict those events. In [30], the analysis studied the effect of system failures, fatal events in particular, on job failure events. The work in [18] also focused on system failures. In cloud environments, limited information is available for monitoring the physical system. Our analysis abstracts the physical system, and correlated execution resources with job types. The seminal work in [24] studied 9 years of failure logs. Probability distributions were computed for various system measurements. The study provides a basis for future probability estimation studied, but it is harder to apply for online measurements.

The performance of scientific applications have been studied in cloud environments [15], [16], [21], [25], [26], [29]. The main focus of those studies was application running time, and whether cloud computing is suitable for scientific applications. Our work does not focus on benchmarking cloud environment, rather it provides insight on individual jobs and enable correlating low-level execution resources with high-level application failures.

Failure prediction using event correlation has been studied in [11]. Temporal and spatial correlations are proposed using multiple features (system information, utilization, packet count, etc). Different learning models were used including time series analysis, and neural network. The supervised learning approach relies on the availability of large set of failure instances, including hardware, software, CPU, OS, filesystem and memory failures. Collecting these detail information might impose extra processing time, which might increase the running time of the entire workflow.

In summary, our work enables job-based failure prediction that takes into account low-level cloud execution environment. The benefit is two-fold: providing feedback to the application scientist about job failure patterns, and enabling online distributed host-based failure prediction. Although our analysis used offline traces collected from workflow executions, the process was conducted incrementally at each point in time, without full knowledge of the entire dataset.

VII. Conclusion

In this paper we showed how machine learning techniques can be used to analyze the behavior of scientific worfklows in cloud environments. In our analysis we considered the failure and success rates of workflow jobs on individual execution resources and the combined rates across a number of resources.

We showed that the Naive Bayes classifier applied to the execution logs of scientific workflows can accurately predict the failure probability of jobs. For the combined model, which also examines the failure probability of a job on other resources, we showed that in some cases, a job destined for failure can potentially be executed successfully on a different resource. This information can be used as input to a workflow management system while it is making decisions about job scheduling onto the resources.

We conducted the analysis on four real workflow applications. In some cases (Periodigrams) we found insufficient failures to conduct the analysis. In some applications (Epigenome and Broadband) our analysis indicated that the workflow failure was more likely due to the problems with the application software (particular workflow components or application environment) rather than particular resource. For robust applications such as Montage, we found that workflow failures were due to data management components used by the Pegasus workflow management system.

Although in this paper we conducted the analysis on Amazon EC2, the methodology is applicable to other clouds and any distributed system: grids, Condor pools, etc. In the future, we plan to evaluate the approach in these environments as well. Further future work also involves integrating the online analysis with Pegasus and evaluating the performance of the analysis and its accuracy in real-time settings.

Acknowledgment

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract DE-AC02-05CH11231. Additional support was provided by NSF grant OCI-0943705.

REFERENCES

[1] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/.

[2] Broadband working group. http://scec.usc.edu/research/cme/groups/broadband.

[3] Periodograms. www.ipac.caltech.edu.

[4] R. www.r-project.org.

[5] USC Epigenome Center. epigenome.usc.edu.

[6] Grid logging: Best practices guide. www.cedps.net/index.php/LoggingBestPractices, 2008.

[7] G.B. Berriman, E. Deelman, J. Good, J. Jacob, D.S. Katz, C. Kesselman, A. Laity, T.A. Prince, G. Singh, and M.-H. Su. Montage: A grid enabled engine for delivering custom science-grade mosaics on demand. In *SPIE Conference 5487: Astronomical Telescopes*, 2004.

[8] Emma S. Buneci and Daniel A. Reed. Analysis of application heartbeats: learning structural and temporal features in time series data for identification of performance problems. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 52:1–52:12, Piscataway, NJ, USA, 2008. IEEE Press.

[9] D. A. Cieslak, N. V. Chawla, and D. L. Thain. Troubleshooting thousands of jobs on production grids using data mining techniques. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, GRID '08, pages 217–224, Washington, DC, USA, 2008. IEEE Computer Society.

[10] E. Deelman, G. Singh, M.H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[11] Song Fu and Cheng-Zhong Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 41:1–41:12, New York, NY, USA, 2007. ACM.

[12] Dan Gunter, Taghrid Samak, Ewa Deelman, Christopher H. Brooks, Monte Goode, Gideon Juve, Gaurang Mehta, Priscilla Moraes, Fabio Silva, Martin Swany, and Karan Vahi. Online Workflow Management and Performance Analysis with STAMPEDE. *7th International Conference on Network and Service Management (CNSM 2011)*, 2011.

[13] Dan Gunter and Brian Tierney. Netlogger: A toolkit for distributed system performance tuning and debugging. In *Integrated Network Management, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003)*, volume 246 of *IFIP Conference Proceedings*, pages 97–100. Kluwer, 2003.

[14] Dan Gunter, Brian L. Tierney, Aaron Brown, Martin Swany, John Bresnahan, and Jennifer M. Schopf. Log summarization and anomaly detection for troubleshooting distributed systems. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 226–234, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans. Parallel Distrib. Syst.*, 22:931–945, June 2011.

[16] Gideon Juve and Ewa Deelman. Scientific workflows in the cloud. In M. Cafaro and G. Aloisio, editors, *Grids, Clouds and Virtualization*, pages 71–91. Springer, 2010.

[17] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P. Berman, and Phil Maechling. Data sharing options for scientific workflows on amazon ec2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–9, Washington, DC, USA, 2010. IEEE Computer Society.

[18] Zhiling Lan, Ziming Zheng, and Yawei Li. Toward Automated Anomaly Identification in Large-Scale Systems. *Parallel and Distributed Systems, IEEE Transactions on*, 21(2):174–187, February 2010.

[19] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 583–588, Washington, DC, USA, 2007. IEEE Computer Society.

[20] Noam Palatin, Arie Leizarowitz, Assaf Schuster, and Ran Wolff. Mining for misconfigured machines in grid systems. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 687–692, New York, NY, USA, 2006. ACM.

[21] Lavanya Ramakrishnan, Piotr T. Zbiegel, Scott Campbell, Rick Bradshaw, Richard Shane Canon, Susan Coghlan, Iwona Sakrejda, Narayan Desai, Tina Declerck, and Anping Liu. Magellan: experiences from a science cloud. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 49–58, New York, NY, USA, 2011. ACM.

[22] Taghrid Samak, Dan Gunter, Ewa Deelman, Gideon Juve, Gaurang Mehta, Fabio Silva, and Karan Vahi. Online Fault and Anomaly Detection for Large-Scale Scientific Workflows. In *13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011)*, Banff, Alberta, Canada, September 2011. IEEE, IEEE Computer Society.

[23] Taghrid Samak, Dan Gunter, Monte Goode, Ewa Deelman, Gaurang Mehta, Fabio Silva, and Karan Vahi. Failure Prediction and Localization in Large Scientific Workflows. In *The Sixth Workshop on Workflows in Support of Large-Scale Science (WORKS11)*, Seattle, WA, USA, November 2011.

[24] Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.

[25] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 193–204, New York, NY, USA, 2010. ACM.

[26] Jens-Sönke Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge, and Bruce Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 15–24, New York, NY, USA, 2011. ACM.

[27] J.S. Vockler, G. Mehta, Y. Zhao, E. Deelman, and M. Wilde. Kickstarting Remote Applications. In *International Workshop on Grid Computing Environments*, number 0, 2007.

[28] Lingyun Yang, Chuang Liu, Jennifer M. Schopf, and Ian Foster. Anomaly detection and diagnosis in grid environments. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 33:1–33:9, New York, NY, USA, 2007. ACM.

[29] Yan Zhai, Mingliang Liu, Jidong Zhai, Xiaosong Ma, and Wenguang Chen. Cloud versus in-house cluster: evaluating amazon cluster compute instances for running mpi applications. In *State of the Practice Reports*, SC '11, pages 11:1–11:10, New York, NY, USA, 2011. ACM.

[30] Ziming Zheng, Li Yu, Wei Tang, Zhiling Lan, Rinku Gupta, Narayan Desai, Susan Coghlan, and Daniel Buettner. Co-analysis of ras log and job log on blue gene/p. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 840–851, Washington, DC, USA, 2011. IEEE Computer Society.