Experience Report: Anomaly Detection of Cloud Application Operations Using Log and Cloud Metric Correlation Analysis

Mostafa Farshchi[1,2], Jean-Guy Schneider[1], Ingo Weber[2,3], John Grundy[1]
[1]School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia
{mfarshchi, jschneider, jgrundy}@swin.edu.au
[2]Software Systems Research Group, NICTA, Sydney, Australia
ingo.weber@nicta.com.au
[3]University of New South Wales, Sydney, Australia

***Abstract*—*Failure of application operations is one of the main causes of system-wide outages in cloud environments. This particularly applies to DevOps operations, such as backup, redeployment, upgrade, customized scaling, and migration that are exposed to frequent interference from other concurrent operations, configuration changes, and resources failure. However, current practices fail to provide a reliable assurance of correct execution of these kinds of operations. In this paper, we present an approach to address this problem that adopts a regression-based analysis technique to find the correlation between an operation's activity logs and the operation activity's effect on cloud resources. The correlation model is then used to derive assertion specifications, which can be used for runtime verification of running operations and their impact on resources. We evaluated our proposed approach on Amazon EC2 with 22 rounds of rolling upgrade operations while other types of operations were running and random faults were injected. Our experiment shows that our approach successfully managed to raise alarms for 115 random injected faults, with a precision of 92.3%.*

Keywords—Cloud application operations; DevOps; Cloud monitoring; anomaly detection; error detection; log analysis.

I. Introduction

Several industry surveys show severe figures of loss of money, market share, and reputation due to various types of system downtime. According to a recent survey (Nov-Dec 2014) reported by International Data Corporation (IDC) [1], an average cost of unplanned downtime in fortune-1000 companies is $100k per hour. This observation is in line with other industry estimates from Gartner [2], Avaya [3], Veeam [4], and Ponemon [5]. These industrial surveys and cost estimation analysis show an hourly cost of applications downtime between $100K and $540k per hour. A separate survey [6] from 205 medium to large business firms in North America states that companies are losing as much as $100 million per year as the result of the server, application, and network downtime. Such immense financial and non-financial losses demonstrate the importance of tackling the root causes of system failures. Operation and configuration issues have been reported to be one of the main causes of overall system failure [7-9]. One recent empirical study reports that operational activities are the root cause of 69% of system-wide outages [9].

One of the reasons for such high percentages of operational failure issues is the complexity of modern large-scale applications, especially in a cloud environment. Cloud environments are inherently complex due to the flexibility provided and the large number of resources involved. Applications in a cloud environment are subject to constant changes from sporadic operations, such as on-demand scaling, upgrade, migration, and reconfiguration [10]. In the maintenance of large-scale applications, several administrators perform tasks and execute various operations. Sporadic operations are usually implemented by a set of separate tools and are subject to interference from simultaneous operations dealing with the same resources. Executing an operation in such an environment is error-prone, as changes to one resource by one operation might affect the correct execution of other operations. With these complexities, it is not surprising that operational-related failures have been reported as one of the main challenges in system failure and outages [10, 11]. However, operation and configuration activities did not receive much of the attention they deserved until the recent movement of DevOps.

One way to improve systems reliability is to leverage a suitable set of tools and techniques to monitor running operations, and to verify their impact on a system in real-time. Unfortunately, most of the past approaches to systems monitoring solely focused on point data [12] which is done by observing the state of hardware and software metrics, such as CPU utilization, network traffic, number of live sessions, and so on. This was done without monitoring the *behavior* of applications operations and inspecting the reflection of the flow of actions in systems resources. In current practice of monitoring cloud application operations, an operation's log is the main source of information for monitoring the operation behavior. Yet, there are several severe limitations in log analysis [13]. Logs are usually low-level, noisy, and they lack information of changes to resource states. These limitations exacerbate the problem of monitoring operations with logs.

978-1-5090-0406-5/15/$31.00 ©2015 IEEE

These challenges make dependability assurance of running operations a very difficult task. To address these difficulties, we have developed a novel and effective approach that adopts a regression-based technique to identify the correlation between states of resources with application operations behaviors. The derived correlation model is then leveraged to verify that the actual state of the system corresponds to the expected state of the system based on operation behavior at runtime. Assertions are used to check if the actual state of the system corresponds to the expected state of the system. Assertion checking is a crucial part of error detection and error diagnosis in monitoring operation process execution. Our approach improves dependability assurance of cloud application operations through the following novel contributions:

- Identifying log events that cause changes to cloud resources by:
  - Clustering low-granular logs using weight timing and correlation coefficients;
  - Applying a regression-based statistical technique to learn correlation and causation relationships between operation behavior and cloud metric changes.
- Defining assertion specifications as predictors of the expected behavior for system operations, based on the correlation and causation model.
- Evaluating the approach on realistic data sets, where we learn from error-free traces, specify assertions, and evaluate if the assertions can detect injected faults.

To evaluate our approach, we collected experimental data from case studies of rolling upgrade operations. Upgrade operations have been highlighted to be a high-risk and an error-prone activity, and yet are a frequent task for systems maintenance and DevOps continuous deployment practices [7]. To obtain data that reflects a realistic environment in the public cloud, all the operations were run on Amazon EC2 using tools like Netflix Asgard and CloudWatch. We ran 22 rounds of rolling upgrade operations while faults were injected at random, and multiple concurrent operations were running. The evaluation of our approach on this data shows promising results.

We first give a background to this work in Section II, followed by a motivational example as our case study in Section III. Then, an overview of our new approach is given in Section IV. Next, the details of steps of the approach are explained in Section V. In Section VI, we present our results, and evaluate our findings. In section VII, key related work is discussed, and finally conclusions and future work are given.

II. Background

In this section we outline background knowledge of sporadic cloud operations, discuss why cloud DevOps operations commonly fail, and highlight limitations of current approaches to operations monitoring through log analysis.

A. Cloud DevOps (Sporadic) Operations

The focus of our work is on monitoring and dependability assurance of DevOps applications operations in public cloud environments, also referred to as "*sporadic operations*". Some types of such sporadic operations are Backup, (Rolling) Upgrade, Cloud migration, Reconfiguration, On-demand scaling, Rollback / Undo, and Deployment. "There is a sporadic nature to these operations, as some are triggered by ad hoc bug fixing and feature delivery while others are triggered periodically." [14]. Sporadic operations are very sensitive as they often have a system-wide impact. In addition, these operations are subject to interference from simultaneous operations, whether through automatic concurrent operations or manual changes applied to a system and its resources.

B. VM Instance Failure

Public cloud computing services, like Amazon Web Services (AWS), are designed and engineered in a way to be fault-tolerant for service delivery. This resiliency is achieved mainly through shared resources, in which the failure of one resource will not significantly affect the whole system. However, it does not mean that all cloud services are fault-tolerant. In fact, many of these services are fault-tolerant to the extent that a cloud customer chooses to architect them.

In contrast to service delivery in the cloud where the status of VM instances is important in an aggregated form, at the operational level (e.g., rolling upgrade, deployment, or backup) each individual instance and its attached resources can be important. From time to time an instance fails – e.g., an instance can freeze or crash and become unresponsive. These failures are usually caused by one of the following: a problem stemming from the resources that the instance is running on; memory over-usage due to increase of system load; an application bug that stresses the instance; an operating system kernel bug; or through random system termination for assessing of systems resiliency and recoverability in production (e.g. Chaos Monkey[1]) [15, 16]. The occurrence of any of these failures during an upgrade can put the upgrade process on hold or derail it; hence, it is important to adopt a mechanism to track and trace the successful execution of an operation.

C. Concurrent Operations and Configuration Changes

Changes in cloud configuration are one of the reasons that make operation validation in this environment very challenging. A recent Gartner report states that over half of the outages of mission-critical systems are caused by "change, configuration, release integration and handoff issues" [11]. A cloud provides a configurable and scalable resource sharing environment, and thus software applications and services are exposed to frequent configuration changes, due to efficient and cost-effective use of these shared resources.

Examples of frequent configuration changes are: detaching or attaching an Elastic Block Storage (EBL) disk volume from / to an instance; changes in conditions and configuration of an

[1] Chaos Monkey is a service that was developed by Netflix to test the resiliency and recoverability of applications running on AWS. Chaos Monkey's job is to randomly kill instances within their group of systems.

Auto Scaling Group (ASG)[2], e.g., its size (horizontal scaling in/out); manual termination or reboot of an instance; instance manipulation for testing purposes; migration of machines to different zones or regions; or changing from one machine type to another (vertical scaling up/down). Such configuration changes of cloud resources may happen frequently. They are another motivation for our work, showing that the validation of sporadic operations has critical importance.

D. Log Analysis

There are several limitations and challenges in systems monitoring from logs. First, logs are often low-level, noisy, and with inconsistencies in style [13, 17]. Many of the current practices of generating logs focus on developer needs during development time, rather than considering administrative needs in production [9]. Second, logs are voluminous, and it is usually difficult to derive which log line, or which set of log lines, is responsible for an action in changing a state of a system resource. In addition, the granularity level of log data is usually different from resource metric data, and this uneven granularity level makes the mapping between these two more challenging. Third, monitoring execution behavior of an operation solely based on the operations' log is not adequate. In large-scale applications, in which hundreds of shared resources are involved, resources are exposed to changes from multiple concurrent operations from time to time. Thus, it is not trivial to isolate the execution of one such operation from other running operations. To tackle these limitations, this study attempts to leverage cloud metric data to cross-validate the execution of cloud DevOps operations.

III. Motivating Example - Rolling Upgrade Case Study

Our study aims to investigate whether it is possible to derive a strong correlation model between event logs of operations and the observable metrics of cloud resources. To conduct this investigation, we chose rolling upgrade, as implemented by Netflix Asgard[3] on top of Amazon Elastic Computing Cloud (EC2), as a case study of such an operation.

A rolling upgrade operation is a good example of a sporadic cloud operation that incurs interference from other operations. Applications in the cloud are deployed on a collection of virtual machines (VMs). Once there is a new version of the application released, a new virtual machine image is prepared with the new version – this is also called "baking the image". Then all the current virtual machines will be replaced by newly baked image through an upgrade process, such as rolling upgrade. A rolling upgrade replaces VM instances, *x* at a time – e.g., upgrading 400 instances in total by upgrading 10 instances concurrently at any given time during the operation. Asgard upgrades each EC2 instance through the following main steps: remove and deregister the instance from Elastic Load Balancer (ELB), terminate the instance, wait until the auto-scaling group replaces the missing instance with a new instance (running the updated version of the application); the new instance is registered with the ELB. There is a chance that an instance faces a configuration change or a failure at any time during these steps.

The contemporary practice of *continuous deployment* focuses on pushing every commit into production – as long as it passes a large number of tests. In such an environment, upgrades can occur with high frequency – between few times a week [18] to many times per day [19], updating hundreds of machines, without causing any service downtime. Therefore, we felt the rolling upgrade was an excellent exemplar target operation for our investigation in this study.

IV. Overview

Our new approach uses a statistical technique to extract a regression-based model that explains the correlation and potentially causalities between operation event logs and cloud resource metrics. The output of the model is used to generate assertions, which are then leveraged for anomaly detection of run-time execution of cloud application operations.

To derive assertions from observations, we assume that there is a stream of time-stamped events, such as events represented by log lines, and at least one cloud platform metric that can be observed. Fig. 1 gives an overview of the approach.

Logs represent the behavior of an operation while metrics show the status of a system. As can be seen in Fig. 1, event logs are generated at various points in time, and metrics are collected at potentially different points in time. Collecting monitoring data points usually happens at fixed intervals, such as every minute or every 5 minutes in Amazon CloudWatch. In contrast, observation of system operations behavior through event logs happens at non-fixed intervals, such as the occurrences of few event logs within one second, and then the absence of any event logs for the next few seconds or even minutes.

Besides directly observed metrics, there can also be derived metrics, such as the difference between the previous and the current data point, or the *n-th* derivative. Furthermore, the observation component can *pull metrics* explicitly, e.g., through API calls. This can be done at fixed intervals, in which case the result is very similar to regularly collected metrics, or pull requests can be issued whenever an event was observed.

This study explores the relationship between the behavior

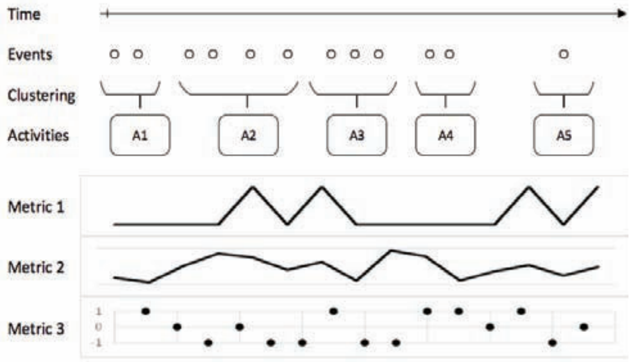


Fig. 1 Assertion derivation from log and metric observations.
Note: A1..A5 are activities; Metric1..3 are metrics like CPU utilization, network usage, number of instances changes etc.

[2] Auto Scaling allows automatic scaling in or out, i.e., de/increasing the number Amazon EC2 VM instances automatically according to conditions defined by a customer.
[3] https://github.com/Netflix/asgard

of application operations and a cloud's resource states. Our research investigates whether adopting a log analysis technique along with a regression-based technique is practical to model the relationship between cloud operation behavior and the changing states of cloud resources. The outcome of this effort is used for detecting anomalies that are happening during the execution of sporadic cloud operations. The high level steps of the approach, also shown in Fig. 2, are as follows: 1) data collection and data metrics derivation; 2) logs-metrics data mapping; 3) Logs clustering; 4) Correlation derivation between logs and metrics 5) Assertion specification for anomaly detection. We describe these steps in detail in the next section.

V. Approach to Assertion Derivation from Statistical Observations

In this section, we give the details of the key steps in our approach, as outlined above.

A. *Data Collection and Data Metrics Derivation*

To set up an experiment and obtain data from a realistic environment, we collected data by running rolling upgrade operations in a public cloud environment. To this end, we used environments and tools that are in wide-spread use in industry: clusters of VMs on Amazon EC2, grouped into Auto Scaling Groups (ASGs), Amazon CloudWatch for collecting cloud monitoring metrics, and Netflix Asgard for executing the operations and collecting event logs.

1) *Metrics from CloudWatch*

CloudWatch provides monitoring metrics for many Amazon services, including EC2, ASG, ELB, etc. In our experiment, CloudWatch is used to collect metrics data for these three services. CloudWatch can collect this data at most once per minute. The data are available in JSON file format and can be retrieved through an API. By default, several metrics related to individual VM instances can be collected by CloudWatch, including CPU utilization, network traffic (incoming / outgoing), failed health checks, and so on. For a whole ASG, averages can be obtained. The data can be used for purposes like monitoring the health of the system or for custom auto scaling.

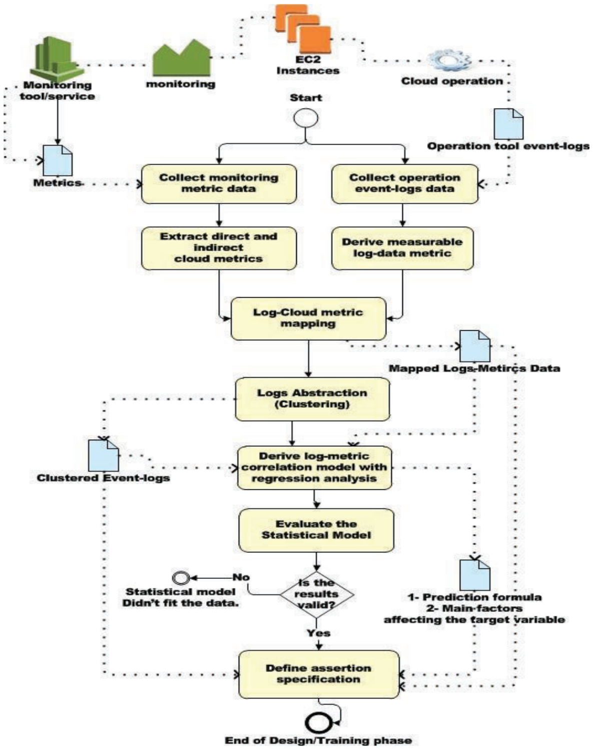

Monitoring tool/service
monitoring
EC2 Instances
Cloud operation
Start
Operation tool event-logs
Metrics
Collect monitoring metric data
Collect operation event-logs data
Extract direct and indirect cloud metrics
Derive measurable log-data metric
Log-Cloud metric mapping
Mapped Logs-Metircs Data
Logs Abstraction (Clustering)
Clustered Event-logs
Derive log-metric correlation model with regression analysis
Evaluate the Statistical Model
No
Statistical model Didn't fit the data.
Is the results valid?
1- Prediction formula
2- Main-factors affecting the target variable
Yes
Define assertion specification
End of Design/Training phase

Fig. 2 Workflow of extracting correlation between logs and metrics

In our study, to verify the effect of logged events, we are interested in the metrics that represent the transitions between states of a VM instance. For example, once a termination action is triggered through operation execution, one VM instance should transition from state "running" to "shutting-down" and eventually to state "terminated". Fig. 3 shows the instance lifecycle of an Amazon EC2 instance.

Interestingly, the metrics available from CloudWatch directly include no metric that explicitly shows the number of instances started or terminated within an ASG. The total number of healthy machines is only partly indicative of that: if, during any minute, a new machine becomes active, and an old one is terminated directly after, the total number of healthy machines remains static. This is a very common occurrence during rolling upgrade. However, the data points for individual instances, like CPU utilization, is only present when the machine is active. We derived precise metrics for the numbers of started and terminated instances, respectively. These metrics derived from CloudWatch data are a cornerstone in our approach, insofar as the case of rolling upgrade is concerned.

2) *Event logs from Operations Tools*

To draw any mapping between cloud metrics and information from operation log lines in textual form, we needed to extract a set of metrics that show the occurrences of different event logs. Although the styles of logging might be different, almost all type of logs contain time-stamped information, whether they are application logs, database logs, or operation logs. Furthermore, log message represent information about an event, including logs that indicate preparation or waiting periods. Hence, we assume a logged event must have at least two attributes: a timestamp and an event description.

In our case study, we used Netflix Asgard to execute rolling upgrade operations and to collect operation logs. Asgard is an open-source web-based tool published by Netflix for managing

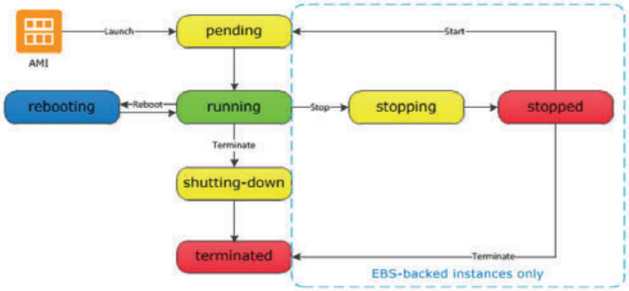


Fig. 3 Amazon EC2 instance lifecycle- source: AWS documentation.

cloud-based applications and infrastructure. Asgard automates some of the AWS cloud operations such as deployment and upgrade. Asgard was developed by Netflix, AWS's largest customer, to provide a higher-level management interface, and since it has been released publicly is in wide-spread use. Its log fulfils our base assumption, and contains high-quality textual messages – albeit the latter is not required in our approach.

In programs with any form of repetition it is common to have logged events of recurring event types. In our case study, every time a VM is terminated, the same type of event is logged, where only certain parameters (VM ID, timestamp, etc.) differ. As part of our data transformation of event logs, all unique types of event logs for an operation are identified. For each event type, similar to the regular expression generation described in [20], we extract a regular expression that matches exactly the log lines belonging to this type [20]. During log processing, the event type for each log line can be identified by matching the log line to the regular expressions. We can then derive a metric that shows the occurrence of each event type over time, throughout the operation process. Fig. 4 shows the pattern of occurrences of the event logs throughout the process of rolling upgrade operation for updating four virtual machine instances. Looking at the figure, it reveals that there are recurring behavior pattern that happening at different time window. Presenting textual log events to a form of quantitative metrics enables us to visualize the behavior of the system through logs, which itself can be helpful for better understanding of the behavior of application operations.

B. *Mapping Event Logs to Metrics Data*

Amazon CloudWatch offers metrics with a granularity no finer than 1 minute. In contrast, events can be logged with a frequency of split seconds or several minutes, as is in part the case in our case study. Therefore, the log and metric data need to be mapped. Since we can observe actual changes to cloud resources only through the CloudWatch metrics, i.e., no more often than once per minute, we chose to interpolate the *occurrence strength* of event log types that occurred within each minute-long time window to the respective minute. The number of occurrences for each event type is extracted as described above in Section V.A.

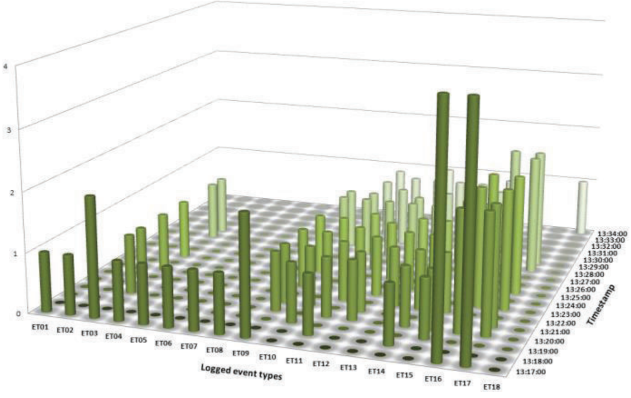


Fig. 4 Visualization of occurrence of 18 event types of rolling upgrade for 4 VM instances

Specifically, the interpolation indicates at which second of a minute an event happened. Indicating a point of time for the derived metric for log events would show a relative interval of happening of set of log events. Therefore, we parse the timestamp of each log message and extract the point of time (seconds of a minute) the event happened. Then a relative occurrence value is calculated as an interpolated value, capturing the time-wise proximity of the event to the full minute before and after the event happened. For instance, say event *E1* happened at *x* minutes and 30 seconds – then its occurrence strength would be counted as 0.5 for both minute *x* and minute *x+1* ; if it happened at *x* minutes and 15 seconds, occurrence strength for minute *x* is 0.75 and 0.25 for minute *x+1* . This interpolated occurrence strength allows us to map the event log data onto the one-minute interval cloud metric data.

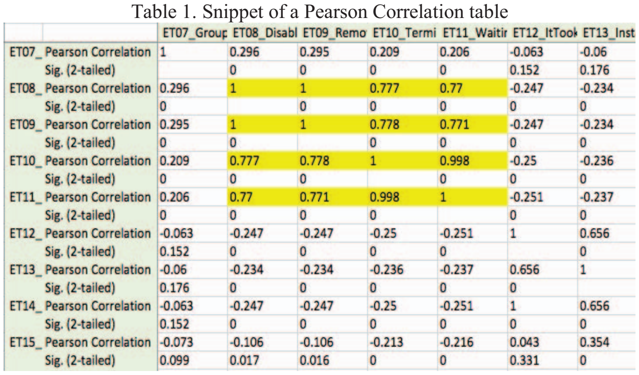
C. *Clustering Logged Events*

We cluster logged events into higher-level activities, for two reasons: (i) Logs are often low-level and voluminous; by raising the level of abstraction, users may find the information provided more useful. (ii) If a set of event types always co-occur, then high correlation among them cause a problem of *multicolinearity* in some statistical models, which might lead to unreliable and unstable estimates.

To facilitate the clustering process, we adopted the *Pearson product-moment correlation coefficient* method to derive a measure of association strength between logged events. Based on the interpolated occurrence strength described above, we use the Pearson correlation coefficient to automatically determine where the strongest association exist between any two event types. Event types with a very high correlation can then be combined into activities.

In our data analysis, we used SPSS to derive the Pearson correlation coefficient between variables by extracting the correlation strength and the direction of the linear association between the types of event logs. Table. 1 shows a snippet of the *Pearson-r* values for correlation distance between 18 different event types of the rolling upgrade operation of upgrading 40 VMs instances, four at a time. The value of *Pearson-r* ranges from -1 to +1; a value of zero or very close to zero indicates that there is no correlation between two variables. A value close to 1 indicates a strong positive correlation between the variables, i.e., in our case, that the events of these types

Table 1. Snippet of a Pearson Correlation table



		ET07_Group	ET08_Disabl	ET09_Remo	ET10_Termi	ET11_Waitir	ET12_ItTook	ET13_Inst
ET07_	Pearson Correlation	1	0.296	0.295	0.209	0.206	-0.063	-0.06
	Sig. (2-tailed)		0	0	0	0	0.152	0.176
ET08_	Pearson Correlation	0.296	1	1	0.777	0.77	-0.247	-0.234
	Sig. (2-tailed)	0		0	0	0	0	0
ET09_	Pearson Correlation	0.295	1	1	0.778	0.771	-0.247	-0.234
	Sig. (2-tailed)	0	0		0	0	0	0
ET10_	Pearson Correlation	0.209	0.777	0.778	1	0.998	-0.25	-0.236
	Sig. (2-tailed)	0	0	0		0	0	0
ET11_	Pearson Correlation	0.206	0.77	0.771	0.998	1	-0.251	-0.237
	Sig. (2-tailed)	0	0	0	0		0	0
ET12_	Pearson Correlation	-0.063	-0.247	-0.247	-0.25	-0.251	1	0.656
	Sig. (2-tailed)	0.152	0	0	0	0		0
ET13_	Pearson Correlation	-0.06	-0.234	-0.234	-0.236	-0.237	0.656	1
	Sig. (2-tailed)	0.176	0	0	0	0	0	
ET14_	Pearson Correlation	-0.063	-0.247	-0.247	-0.25	-0.251	1	0.656
	Sig. (2-tailed)	0.152	0	0	0	0	0	0
ET15_	Pearson Correlation	-0.073	-0.106	-0.106	-0.213	-0.216	0.043	0.354
	Sig. (2-tailed)	0.099	0.017	0.016	0	0	0.331	0

(almost) always co-occur. Negative values indicate that events of the respective event types rarely co-occur. It is important to note that clustering event logs simply based on the correlation of occurrences of the events might not lead to a set of meaningful activities. Where user interaction is a direct application, other factors should be taken into consideration, e.g., as done in [20].

D. Statistical Event-Metric Correlation Derivation

Correlation is defined as a statistical tool to measure the degree or the strength of association between two or multiple variables, whereas causation expresses the cause and effect between variables [21]. It is important to note that while the presence of causation certainly implies correlation, the existence of correlation only implies a potential causation. For instance, one may observe a correlation between power consumption and number of failures; yet the underlying cause of a higher number of failures could be due to the increase chance of observing any failure when a higher number of VMs is involved as the result of scaling up process to respond incoming higher workload traffic.

Correlation is a powerful tool, as it can signify a predictive relationship that can be exploited in practice, especially for forecasting. To infer whether a correlation implies causality, one may need to make sure the correlation is extracted from a controlled environment, i.e., to make sure there are no factors, other than the ones included in the analysis, affecting the target variable. If this criterion is fulfilled, a meaningful correlation can be interpreted as causation.

For the purpose of this study, we are interested to find the effect of operation actions on cloud resource state changes. To this end, we start from data that has been collected for a period of time and has sufficiently many data points. We then use a regression-based technique to discover correlation between logged events and changes in metrics, i.e., the absence, presence, and strength of such changes. In our running example, e.g., whenever there is a log event that indicates a termination request for one VM has been issued, the expectation is that within the next minute one VM will traverse from status running' to 'shutting-down', and finally 'terminated'. As the example suggests, it is rational to assume that there is a direct linear relation between a logged action and its effect as a change of a VM state.

In our analysis, we adopted a Multiple Regression technique, namely Ordinary Least Squares (OLS) regression. “There are two general applications for multiple regression: prediction and explanation” [22]. This means, first, multiple regression can be utilized to predict an outcome for a particular phenomenon, based on the knowledge available from some other correlated variables [23]. Second, multiple regression can be used to understand how much of the variation of the outcome can be explained from the correlated variables. Therefore, multiple regression is done for several independent variables (IV) as predictors, and one dependent variable (DV) as the outcome.

Given y is the DV, $x_1 \dots x_n$ to be the IVs, and ϵ to be the *error term* or *noise,* the general form of the linear regression function is

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

where the intercept α denotes a constant value that is the expected mean value of y when all $x = 0$. The coefficients $\beta_1 \dots \beta_n$ denote the effect of each variable on an overall model. The coefficient parameters measure the individual contribution of independent variables to the prediction of the dependent variable after taking into account the effect of all the independent variables.

Several types of linear regression models are based on the above mechanism, and these types differ in the kinds and distribution of data they are suitable for. One challenge is to find a model that fits the data at hand well. Most of these models can be generated by using standard statistical software packages such as R, SPSS, SAS, STATA, and so on. We analyzed our data with multiple regression and generalized linear regression models, including Poisson regression and Negative Binomial regression. It is beyond the scope of this paper to explain the details result of these models. The multiple regression model (OLS) provided the best fit for our data.

Multiple regression is a robust model, used as the base of data analysis in many disciplines. It is important to note that, in contrast to many common uses of multiple regression, e.g., in the social and medical domain where the sample data collection is expensive, limited, and comes with a degree of bias, for our use in log and metric data analysis there is higher confidence in accuracy of the sample data as they are collected through machine-based observations. Further, in our approach the emphasis is on validation of the regression results through empirical evaluation, rather than mere generalization from the observation of sample data.

E. Assertion Derivation for Fault Detection

Once we discovered correlation and causation relationships between events and metrics, these can be formulated as assertions, such that an assertion evaluation service can determine at runtime if the assertions are fulfilled. If any assertion is violated, the service will raise an alarm, e.g., to trigger automatic diagnosis or remediation actions. To derive assertions, we extract the regression equation from the multiple regression coefficient results, where y denotes the dependent variable (e.g., number of terminated instances), and x_1 and x_3 refer to the *relevant activities*, the regression equation is:

$$y = \alpha + \beta_1 * x_1 + 0 * x_2 + \beta_3 * x_3 + \ldots + 0$$

From the model, we learn concrete values for α and the β_i. In particular, for any x_i where the explanatory anlaysis of correlation is below the threshold, we set $\beta_i = 0$. At runtime, each log event is processed as outlined earlier in this section, so that an interpolated occurrence strength for each of the independent variables (x_1 and x_3 in the example) can be obtained. Every minute, a prediction can be calculated and compared with the actual CloudWatch metrics.

One challenge remains: while some of the CloudWatch metrics are discrete values (such as the number of started / terminated VMs), the prediction always has a continuous result value. This value then needs to be discretized. The easiest method for discretization is rounding the number, but that may lead to prediction with a fairly low confidence, e.g., if the predicted value states that $y = 0.5$ VMs were terminated. Another method is to define a threshold t, such that $0 < t < 0.5$, and the prediction is set to the integer i closest to y iff y is closer to i than t, i.e., $|y - i| < t$. Finding a suitable threshold has to be done for each application scenario separately, as a tradeoff is needed between missing too many real alarms (false negatives) and receiving too many false alarms (false positives).

VI. Experimental Results and Evaluation

In this section, we describe how we applied our approach to the case study, first by learning a model from observations of positive cases, and second by using the learned model for prediction and fault detection in cases where faults were injected.

A. Logs Clustering Learning by Pearson Coefficient

We generated the Pearson correlation coefficient for two different data sets. First, we generated correlation data for running a rolling upgrade of 8 virtual machine instances, upgrading two instances at a time. Then, we defined a rule that event types to be grouped together where they had correlation strength of more than 75% (Pearson-r > 0.75) whereas values show highly statistically significant (P-value <0.01). In other words, as a rule, any event type of activity should indicate at least 75% correlation with any other event types of the group that formed an activity. One may choose a higher or lower level depends on the desired abstraction level to obtain from logs. We chose Pearson-r > 0.75 as it is low enough to avoid multicollinearity in our regression analysis while it is high enough to associate strongly correlated event types together. To make sure that the correlation of activities is not affected by different configuration and scales of the operation, we applied the same process for running rolling upgrade of 40 virtual machine instances, upgrading four instances at a time.

In both experiments, although there were slight changes in correlation values, the log abstraction led to identical clustering results. The 18 event types are grouped into six clusters of event logs (i.e. activities). Note that the whole process of log abstraction was done in an automated manner without relying on domain knowledge. To assess how meaningful our log abstraction result is, we investigated the context of the logs entries; the result, which listed in Table 2, shows that all the event types of each cluster are meaningfully related to each other. For instance, the four events of *DisablingXInELB, RemoveInstanceFromELB, TerminateInstance, and WaitingInstancesPending* that are clustered automatically together are related to the action of terminating a VM instance. For simplicity of the analysis, each cluster was given a name according to the context of its event types. Logs to activity mapping are listed in Table 2.

Further, we compared our result with the log abstraction of the same operation that was reported in a previous study [10], which was extracted based on domain knowledge expertise. In the comparison, we did not find any conflict in term of mapping event logs to activities though the level of abstraction is slightly different in the previous work. Based on the above, we concluded that the derived log abstraction is meaningful and appropriate for further analysis with regression.

Table 2. Event logs to activity abstraction for the rolling upgrade operation

Event (Shorten name derived from event log message)	Activity
ET01_StartedThread ET02_UpdatingLaunchWithAmi ET03_CreateLaunchConfig ET04_UpdatingGroupXToUseLaunchConfig ET05_UpdateASG ET06_SortedInstances ET07_GroupXInstancesWillBeReplacedXAtATime	Start of Rolling upgrade (sorted instances)
ET08_DisablingXInELB ET09_RemoveInstanceFromELB ET10_TerminateInstance ET11_WaitingInstancesPending	Remove Instance from ELB and Terminate Instance
ET12_ItTookXminInstanceToBeReplaced ET13_InstanceInLifeCycleStatePending ET14_WaitingForInstanceToGoInService	Instance Replacement Process
ET15_ItTookXminInstanceToGoInService ET16_WaitingForInstanceToBeReady	New Instance to go in service
ET17_InstanceXIsReady	Instance is ready
ET18_Completed	Rolling upgrade completed

B. Correlation and Causality Learning with Multiple Regression Model

To perform correlation and causation analysis, we used the rolling upgrade operation. Similar to the log-clustering step, we performed our analysis based on two separate case studies of rolling upgrades: first, running the multiple runs of rolling upgrade of 8 instances, upgrading 2 instances at a time; and the other, running rolling upgrade of 40 instances, upgrading 4 instances at a time. While we gave a summary of the overall fitness of the model for the both experiments in the Table 3, for conciseness from here on we focus on the details of the learning approach based on the case of 40 instances.

To investigate the relationship between occurrence of event types and cloud metrics in the regression model, the activities derived from log clustering were assigned as predictor variables and the termination and start of instances as two dependent variables, i.e., in two separate models. A multiple regression was run to predict termination of instances given six activities from 514 records data of 10 rounds of running rolling upgrade operations for upgrading 40 instances in Amazon EC2. These six variables statistically significantly predicted termination of instances, $F(4,\ 509) = 1317.097, p < .0005$, adj. $R^2 = .912$. There was a *very strong, positive,* linear correlation between multiple six activities and terminated instances ($R = 0.955$). All six variables added statistically significantly to the prediction, $p < .05$. The result shows 91.2% of the variation in Terminated Instances can be explained by the linear relationship between above five variables and the Terminated Instances, $R^2 = 0.912$. In other words, the regression model, indicating a very good fit to the model, and explained 91.2% of the variability in the yield data. A similar interpretation of the model can be inferred from Table. 3 for the experiments with 8 instances and 40 instances for both terminated and started instances. As with started instances, we observe less coefficient determination in comparison with terminated instances, yet the values are strongly correlated and statistically significant, which indicates the model is a fairly good fit for the data.

Table 3. Coefficient Determinations of multiple regression analysis

Experiment	Metric	R	R Square	Sig. (*p*)
8 instances	Terminated	.918	.842	.000
8 instances	Started	.827	.683	.000
40 instances	Terminated	.955	.912	.000
40 instances	Started	.886	.786	.000

One of the objectives of doing multiple regression analysis was to find an explanatory relationship between independent variables (activities) and the dependent variables (cloud metric). We seek to distinguish the activities that are likely to affect the target metric from the others. To perform such analyzes, we look at the Coefficient results generated by regression analysis, Table 4.

By looking at *p-value* in Table. 4, we observed that activities A3 and A4 are statistically insignificant (p >.005). Therefore, we conclude that these two variables can not explain the variation of the target variable. Further, Standardized Coefficient shows the contribution of activities 01, 06 are almost zero. Above observation helped us to narrow down the contributed activities to A3 and A5. We run the multiple regression again with these two activities for the outcome as shown in Table 5.

Given the high difference between the activity 03 (0.836) and the activity 05 (0.15), it can be concluded the activity 03 is the main cause of termination of an instance. The context of the activity log confirms that the model was effective to identify correctly the activity log that causes the termination, which these finding can be used to define assertion specification as it was explained in Section V.E.

Table 4. Coefficient Correlation - 40 instances – Terminated Metric

	β	Std. Error	B	*Sig.*
Intercept (Constant)	0.083	0.027	---	0.003
A1_Start of Rolling upgrade	0.529	0.208	0.035	0.011
A2_Terminate Instance	1.139	0.026	0.836	0.000
A3_Instance Replacement	-0.023	0.016	-0.019	0.168
A4_New Instance to go in service	-0.023	0.018	-0.017	0.214
A5_Instance is ready	0.201	0.026	0.15	0.000
A6_Rolling upgrade completed	-0.73	0.184	-0.058	0.00

*Note. β = Unstandardized regression coefficient; B = Standardized regression coefficient; Sig. = *p-value*

Table 5. Coefficient Correlation for identified influential factors - 40 instances - Terminated Metric

	β	Std. Error	B	*Sig.*
Intercept (Constant)	0.051	0.021	---	0.018
A2_Terminate Instance	1.197	0.024	0.879	0.000
A5_Instance is ready	0.149	0.023	0.111	0.000

*Note. β = Unstandardized regression coefficient; B = Standardized regression coefficient; Sig. = *p-value*

C. *Prediction and Fault Detection*

In order to evaluate how well the derived assertions can detect failures, we conducted another experiment which was run separately from the one used to learn the model. The raw data of this experiment was obtained from experiments run by our colleagues [24]. The experiments were conducted on Amazon EC2, upgrading 8 instances, 2 instances at the time. Rolling upgrade was executed while multiple tasks (HTTP loads, CPU intensive tasks, and Network intensive tasks) were running, and faults simulating individual VM failure were randomly injected into the system. We obtained data on 22 rounds of rolling upgrade operations, including 574 minutes of metric data and 5335 lines of logs emitted by Asgard. A total of 115 faults were injected at random.

As explained in Section V.E, the equations derived from multiple regression models can be used to predict the number of started / terminated instances within the last minute: given the observed log lines how many VMs should have been started or terminated? If this predicted value does not match the actual value, an alarm is issued. We aim to find out how accurately our approach can identify the anomalies. Since the injected faults were all VM failure, our approach tries to distinguish between VMs being terminated due to legitimate operational activity and cases caused by fault injection. We choose to inject VM failure because the scope of our work has a focus on DevOps/sporadic operations. For such operations - in particular, for the rolling upgrade case study used in this study - the state of VMs is a prime source of anomalies. Therefore, it was reasonable to generate failure types that cause VM termination rather than other types of failures.

To measure the *Precision* and *Recall* of the prediction we classified the result of the prediction into four categories: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). Table 6 explains these four categories in terms of an alarm being issued (or not), and a fault being injected (or not). For any of the 574 minutes of data, we aim to raise an alarm when a fault was injected (TP) or raise no alarm when no fault was injected (TN). FP and FN thus mark cases where the prediction did not work. These four categories are the basis for calculating precision, recall, and F-measure. Precision is a measure to assess the exactness of the result: the percentage of the valid issued alarms out of all issued alarms, $= \frac{TP}{TP+FP}$. Recall is a measure of completeness of correct alarms: the percentage of injected faults where an alarm was raised, $= \frac{TP}{TP+FN}$. The F-measure is the weighted average (harmonic mean) of precision (P) and recall (R), $F_1 = 2 * \frac{P*R}{P+R}$.

In our study, we observed possible delays between an operation action and its effect becoming observable. For instance, consider the duration of terminating one VM: the time between when the respective event was logged until the VM is actually terminated may vary between fifteen seconds and three minutes. It is thus not uncommon that a VM is terminated in one minute, but CloudWatch metrics only reflect the termination in the next minute, or even later. This delay is observable in legitimate operations' actions, as well as in injected faults. Therefore, we studied the results of applying three different time windows (TW) for prediction: zero minutes

Table 6. Classification metric for the generated alarm

	Fault Injected	Fault Not Injected
Prediction ≠ Actual: Alarm	TP	FP
Prediction = Actual: No Alarm	FN	TN

(0mTW), i.e., only the current minute; one minute distance (1mTW), i.e., the current minute, the minute before, and the minute after; and two minutes distance (2mTW), i.e., from two minutes before to two minutes after. It should be noted that a longer TW also delays when the result of the prediction becomes available. This is an application-specific trade-off in practice: is it worth waiting two minutes longer for an alarm, if the precision goes up by x percent? The results of monitoring the operation with the three different time windows are shown in Table 7: Precision, Recall, and F-Score are given without considering the impact of the ripple effect of the faults. Faults are injected randomly on the VM instances, and the occurrences of the faults are logged. However, faults may have ripple effects that lead to seemingly false alarms at a later stage of the operations process, as explained below. As can be seen from Table 7, there are strong differences between the basic precision value of 0mTW and 1mTW of 0.145 (14.5%). The difference between 1mTW and 2mTW, in contrast, is rather small. This observation can be explained because of the time delay that the action of termination takes to be completed: the majority of terminations is completed either within the current minute or the next minute – it rarely takes more than that. Time window size for alarms can be configurable in a real-time monitoring system. For our experiment, we concluded that 1mTW offers a good trade-off between capturing most anomalies and keeping the delay short.

Not all the effects of injected faults are seen immediately. There are cases where failures have *ripple effects*. Table 8 shows three types of ripple effects we observed in the experiment, as well as their number of occurrences. The first two types are essentially race conditions when rolling upgrade and fault injection both want to terminate a particular VM. In particular, rolling upgrade retrieves the list of VMs to be replaced at the beginning of the process, and subsequently goes through the list and attempts to terminate instances. If a VM has already been terminated earlier by fault injection – whether or not correctly detected by our approach at that time – this is not taken into account by Asgard. Instead, the log states that Asgard attempted terminating a VM, and no such effect is observed – hence an alarm is raised. Since no fault had been injected at that time, the alarm is counted as FP in the basic detection, cf. Table 7. To distinguish actual failed prediction from ripple effects (where the prediction behaved as expected), we analyzed all 30 FP and 7 FN cases for the chosen 1mTW (in total 37 cases), by looking at details of the log lines and metrics. We found that 28 out of 37 FN/FP cases were caused by ripple effects of fault injection. Since the prediction behaved as expected in these cases, we re-classified them as TP/TN, leading to the final results shown in Table 9 and in Fig. 5.

Table 7. Evaluation results - basic detection

Evaluation Metrics	0mTW	1mTW	2mTW
Precision	0.567	0.712	0.745
Recall	0.670	0.914	0.921
F-Score	0.706	0.826	0.849

Table 8. Type of ripple effects observed in the experiment

Occurrences	Ripple Effect Explanation
21	Rolling upgrade's attempt to terminate a VM has no effect, since the respective VM has already been terminated by fault injection.
5	Fault injection's termination attempt fails due to instance being already terminated by rolling upgrade earlier.
2	Instance is terminated by fault injection while being pending to be started.

Table 9. Evaluation results – detection result with ripple effect

Evaluation Metrics	1mTW	1mTW_with Ripple Effect
Precision	0.712	0.923
Recall	0.914	1.000
F-Score	0.826	0.960

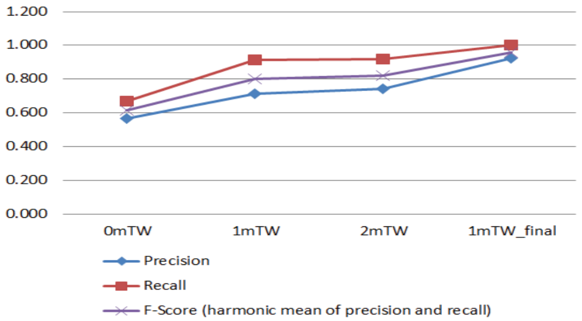


Fig. 5 Showing the Precision, Recall, and F-Score for three time window and for the investigated cases for 1mTW

VII. Related Work

There are two main areas of work related to our approach: error detection and error diagnosis through log analysis, and anomaly detection through system resource health and performance monitoring.

System error detection and diagnosis is an effort that relies on relationship analysis of a deficiency in a system and its observable symptoms. In recent years, several studies have been conducted to automate this process. Kavulya et al. [25] have categorized automated techniques based on mapping the relationship between systems symptoms and failure. These techniques include rule-based techniques, model-based techniques, statistical techniques, machine-learning techniques, count-and-threshold techniques, and visualization techniques [12, 25]. Adopting each of these techniques comes with limitations. For example, rule-based techniques require large knowledge bases that are difficult to maintain; model-based techniques require a detailed understanding of the system. Statistical techniques have been widely used for anomaly detection in system monitoring [12, 26, 27]. Most of the existing work in this domain focus on changes in non-contextual data points (CPU utilization, memory usages, and etc.) and raise an alarm when there are breaches of thresholds. In our research we were not interested in seeing the non-state-based metrics like CPU utilization as we are specifically interested in anomaly detection during running DevOps operations. For anomaly detection during such sporadic operations, state based metrics like VM start / termination are most suitable. Existing literature is very rich in anomaly detection with data points; however, there have been few studies on using contextual and behavioral information for anomaly detection [12]. In this direction, our research focuses on using contextual logs as one of the sources of information for anomaly detection along with metrics data.

Most past approaches that use contextual information are appropriate for offline assessment, rather than for online assessment. Other approaches are mostly intrusive, i.e., they require changes to be applied to the system. POD-Monitor [24] attempted to address this gap by using contextual logs and data point metrics to suppress false alarms of detected anomalies in resources usage. However, their approach lacks the support for anomaly detection of steps of cloud operations that are proposed in this paper. Their paper considers operational context on the level of whole operation processes and focus on anomaly detection on resources, whereas we conduct anomaly detection at fine-grained level of individual steps of operations. Another approach that addresses the above limitations, in part of one of the author's previous work, called *POD-Diagnosis* [10]. This approach models the cloud sporadic operations as processes and uses the process context to catch errors, filter logs and perform on-demand assertion checking for online error handling [10, 28]. This technique addresses the problem of online validation of operations to some degree. However, the approach has two limitations, which we discuss below: it merely relies on logs as the only source of information, and it requires manual assertion specification.

The first limitation of using system operation logs as the main source of information for error diagnosis suffers from logs often being low-level, noisy, and voluminous and with inconsistencies in style [13, 29]. These limitations exacerbate the difficulty of validation of cloud operations. These problems make the validation of processes merely with log analysis less reliable. Therefore, it is important to employ one or more additional sources of information along with the information extracted from logs for validation of running operations. The second limitation is related to manual assertion specification. Assertions check if the actual state of a system corresponds to the expected state of a system. In the previous work [10], intermediate expected outcomes of process steps have been defined manually as assertions. This method is suboptimal for the following reasons. First, manual assertion specification is very time-consuming and thus, with fast evolving changes of modern applications, might not be practical. Second, manual assertion checking might not correctly correlate the changes of operation state with changes to cloud resources. Therefore, there may be a lack of precision in the assertion specification.

Third, manual assertion specification relies on the domain knowledge of the administrator specifying the assertion. This knowledge may be incomplete, and its encoding in assertions may be incomplete. For instance, for a 10-step process touching on 20 resources with an average of 10 parameters each, a full specification of all desired and undesired changes results in 10x20x10=2000 potential assertions. It is unlikely that any administrator will (correctly) specify all of them. This will result in a partial coverage of assertions, potentially leaving out important causes for failures simply because the administrator has never experienced them. Our approach differs from these approaches as we rely on statistical correlation analysis rather than domain knowledge.

System monitoring is of paramount importance for both cloud service providers and cloud service consumers. Analytical tools in cloud monitoring can be used for real-time performance monitoring to quickly uncover performance bottlenecks and troubleshooting unknown issues. Monitoring data can be collected through automatic calls of APIs in near real time fashion. This capability provides a significant opportunity to leverage such data for anomaly detection. Many commercial and open source platforms and services are available for cloud monitoring, including CloudWatch, AzureWatch, CloudKick, Nagios, and OpenNebula. A detailed comparison of monitoring platforms and services is given in [30]. One of the highlighted issues in this domain is the lack of *cross-layer monitoring* [30]. Cross-layer monitoring is a challenging task, as it is difficult to map two different monitoring data types and to interpret them in an integrated form. Our research, in particular, contributes in this direction, as we consider two different sources of monitoring information.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have addressed the problem of monitoring cloud application operations through log and metrics analysis. Our contribution is a novel approach that assists in the reliable assurance of correct execution of sporadic cloud operations as are common practice in DevOps, especially the staged upgrade of running VMs. Core to this approach is a regression-based correlation analysis technique that identifies the correlation between event logs of operations and cloud resource changes, respectively. We showed that the derived regression model can be used as the basis for generating runtime assertions in order to detect anomalies in running operations. We evaluated our approach on the Amazon public cloud computing service (EC2) where, multiple operations were running and random faults were injected. Our results demonstrate that our regression-based analysis technique was able to detect injected faults with high precision and recall.

We aim to use the proposed approach for better error diagnosis. Furthermore, we plan to utilize this approach for designing self-adaptive operations. Such a self-adaptive operation would be able to perform self-healing actions after a failure happens, as well as utilize its knowledge for adapting the configuration of itself, other operations, or the affected application(s) in certain cases like spikes in the demand.

IX. ACKNOWLEDGMENTS

This work is supported in part by the National ICT Australia and Swinburne University of Technology. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

[1] S. Elliot, "DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified," International Data Corporation (IDC), , Dec 2014.

[2] D. Cappuccio, "Ensure cost balances out with risk in high-availability data centers," Gartner, July 2013.

[3] Avaya. (2014,Access: May 2015). *Network downtime results in job, revenue loss*. Available: http://www.avaya.com/usa/about-avaya/newsroom/news-releases/2014/pr-140305/

[4] Veeam, "Veeam data centre availability report 2014," Veeam Software, December 2014.

[5] Ponemon, "Breaking down the cost implications of a data center outage," Ponemon Institute, , December 2013.

[6] Infonetics, "The cost of server, application, and network downtime," Infonetics Research, Janaury 2015.

[7] O. Crameri, N. Knezevic, D. Kostic, R. Bianchini, and W. Zwaenepoel, "Staged deployment in mirage, an integrated software upgrade testing and distribution system," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, Stevenson, Washington, USA, 2007, pp. 221-236.

[8] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do*, et al.*, "What bugs live in the cloud?: A study of 3000+ issues in cloud systems," presented at the Proc. of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 2014.

[9] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao*, et al.*, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems," in *Proc. of the 11th USENIX conference on Operating Systems Design and Implementation*, Broomfield, CO, 2014, pp. 249-265.

[10] X. S. Xu, L. Zhu, I. Weber, L. Bass, and W. Sun, "Pod-diagnosis: Error diagnosis of sporadic operations on cloud applications," in *The 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014.

[11] R. J. Colville and G. Spafford. (2013, Access: Sep 2014). *Configuration management for virtual and cloud infrastructures*. Available: http://goo.gl/C2d4Sz

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR),* vol. 41, p. 15, 2009.

[13] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM,* vol. 55, pp. 55-61, 2012.

[14] L. Z. Xiwei Xu, Daniel Sun, An Binh Tran, Ingo Weber, Min Fu, Len Bass, "Error diagnosis of cloud application operation using bayesian networks and online optimisation " in *11th European Dependable Computing Conference (EDCC)*, Paris, France, 2015.

[15] J. Atwood. (2011,Access: Sep 2014). *Working with the chaos monkey*. Available: http://blog.codinghorror.com/working-with-the-chaos-monkey/

[16] Netflix. (Feb 2014,Access: Sep 2014). *Chaos monkey*. Available: https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

[17] A. Oliner, A. Ganapathi, and W. Xu. (2011) Advances and challenges in log analysis. *Queue*. 30-40.

[18] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at facebook," *IEEE Internet Computing,* vol. 17, pp. 8-17, 2013.

[19] J. Miranda. (2014,Access: June 2015). *How Etsy deploys more than 50 times a day*. Available: http://www.infoq.com/news/2014/03/etsy-deploy-50-times-a-day

[20] Ingo Weber, Chao Li, Len Bass, Xiwei Xu, and L. Zhu, "Discovering and visualizing operations processes with pod-discovery and pod-viz," in *The 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Rio de Janeiro, Brazil, 2015.

[21] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and practice*: OTexts, 2014.

[22] J. W. Osborne, "Prediction in multiple regression," *Practical Assessment, Research & Evaluation,* vol. 7, pp. 1-9, 2000.

[23] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis* vol. 821: John Wiley & Sons, 2012.

[24] X. Xu, L. Zhu, M. Fu, D. Sun, A. B. Tran*, et al.*, "Crying wolf and meaning it: Reducing false alarms in monitoring of sporadic operations through pod-monitor," in *First International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS2015)*, Firenze, Italy, 2015.

[25] S. Kavulya, K. Joshi, F. Giandomenico, and P. Narasimhan, "Failure diagnosis of complex systems," in *Resilience assessment and evaluation of computing systems*, K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 239-261.

[26] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks,* vol. 51, pp. 3448-3470, 8/22/ 2007.

[27] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield*, et al.*, "Statistical techniques for online anomaly detection in data centers," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, 2011, pp. 385-392.

[28] X. Xu, I. Weber, L. Bass, L. Zhu, H. Wada*, et al.*, "Detecting cloud provisioning errors using an annotated process model," in *Proc. of the 8th Workshop on Middleware for Next Generation Internet Computing*, 2013, p. 5.

[29] L. Bass, I. Weber, and L. Zhu, *Devops: A software architect's perspective*: Addison-Wesley, 2015.

[30] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks,* vol. 57, pp. 2093-2115, 2013.