Metric Selection and Anomaly Detection for Cloud Operations using Log and Metric Correlation Analysis

Mostafa Farshchi[a,b], Jean-Guy Schneider[a], Ingo Weber[b,c], John Grundy[d]

[a]*School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia*
[b]*Data61, CSIRO, Australia*
[c]*University of New South Wales, Sydney, Australia*
[d]*School of Information Technology, Deakin University, Melbourne*

Abstract

Cloud computing systems provide the capabilities to make application services resilient against failures of individual computing resources. However, resiliency is typically limited by a cloud consumer's use and operation of cloud resources. In particular, system operation failures have been reported as one of the leading causes of system-wide outages. This applies specifically to DevOps operations, such as backup, redeployment, upgrade, customized scaling, and migration – which are executed at much higher frequencies now than a decade ago. We address this problem by proposing a novel approach to detect errors in the execution of these kinds of system operations, in particular focusing on rolling upgrade operations. Our regression-based approach leverages the correlation between operations' activity logs and the effect of operation activities on cloud resources. First, we present a metric selection approach based on regression analysis. Second, the output of a regression model of selected metrics is used to derive assertion specifications, which can be used for runtime verification of running operations. We have conducted a set of experiments with different configurations of an upgrade operation on Amazon Web Services, with and without randomly injected faults to demonstrate the utility of our new approach.

Keywords: Cloud application operations; cloud monitoring; metric selection, anomaly detection; error detection; log analysis.

1. INTRODUCTION

The failure of systems and software has been the topic of research and investigation for a very long time. Despite many advances in this area failure rates are still high. Several industry surveys show significant loss of money, market share, and reputation due to various types of system downtime. According to a survey conducted by International Data Corporation (IDC) [1] in late 2014, the 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 surveys estimate the cost of application downtime between $100K and $540K per hour. A separate survey from 205 medium to large business firms in North America states that companies are losing as much as $100 million per year as a result of server, application, and network downtime, respectively [6]. Such significant losses (both in monatery and non-monetery terms) demonstrate the need to address the key reasons for system failures. Operation and configuration issues have been reported to be one of the main causes of overall system failure [7, 8, 9], and an empirical study conducted by Yuan et al. [9] reports that *operational activities* are the root cause of up to 69% of system-wide outages.

One of the reasons for such high percentages of operational failure issues is the complexity of modern, large-scale applications, especially in cloud environments. These environments are inherently complex due to the flexibility provided and the large number of resources involved. Applications in a cloud environment are subject to regular changes from sporadic operations, such as on-demand scaling, upgrade, migration, and/or reconfiguration [10]. 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 like Rolling Upgrade in such an environment is error-prone, as changes to one resource (e.g., the state of a VM) may 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 failures and outages [10, 11]. However, operation and configuration activities have not received the much deserved attention until the recent emergence of DevOps.

One way to improve a system's reliability is to leverage

Email addresses: mfarshchi@swin.edu.au (Mostafa Farshchi), jschneider@swin.edu.au (Jean-Guy Schneider), ingo.weber@nicta.com.au (Ingo Weber), j.grundy@deakin.edu.au (John Grundy)

a set of tools and techniques to better monitor running operations and to assess their impact on a system in real-time. In systems monitoring, one problem is that system operators have to deal with tracking multiple monitoring metrics and receive too much monitoring information, including many false warnings and alerts. This distracts system operators from becoming aware of critical abnormal situations [12, 13]. This problem in particular has caused operators to disable monitoring when sporadic operations are running, so as to avoid too many false alerts [14]. Another problem is that most of the existing approaches to systems monitoring solely focused on point data [15]: they observe the state of hardware and software metrics, such as CPU utilization, network traffic, a number of live sessions etc. without monitoring the *contextual behavior* of operations or inspecting the impact of operation steps on systems resources. In current practice of monitoring cloud application operations, an operation's log is the primary source of information for monitoring the *operation's behaviour*. Logs provide valuable information about running operations, yet they are not fully reliable as there are various limitations in log analysis [15]. Logs are usually low-level, noisy, and lack information about changes to resource states. These shortcomings make dependability assurance of running operations a very challenging task.

We have developed a novel, statistical approach to address these difficulties through learning from the correlation between resources and operations' activities. In our approach, we first leverage a statistical technique to cluster low-granular-level logs to a higher abstraction level, in order to form a set of correlated activities. Next, we propose a technique to identify monitoring metrics that are sensitive to an operation's steps, which helps us to form a set of statistically relevant candidate metrics suitable for anomaly detection. We then present an approach using regression analysis to identify which set of operation log events (i.e. activities) affect the target resource metrics, resulting in an assertion specification model. In this process, assertions are used to check if the actual state of the system corresponds to the expected state of the system. This assertion checking is a crucial part of error detection in monitoring the execution of operation steps. Finally, we leverage the selected metrics at runtime, to verify that the actual state of the system observed from monitoring metrics actually corresponds to the expected state of the system based on the assertion specification. In addition, we propose an algorithm to distinguish anomalies that are triggered from the ripple effect symptoms of a failure rather than being caused by the direct effect of a failure.

To assess and evaluate these proposed approaches, we conducted experiments using a public cloud environment of Amazon EC2 by running multiple rounds of rolling upgrade operations. We used Amazon CloudWatch metrics as the source of monitoring resource metrics and Netflix Asgard logs as the source of operation log events.

In summary, our approach improves the dependability assurance of cloud application operations, within the scope of above experimental setting, through the following key contributions:

- Identifying log events that cause changes to cloud resources by:
  - (i) clustering low-granular logs using weight timing and correlation coefficients;
  - (ii) utilizing a regression-based statistical technique to learn correlation and causation relationships between operation behavior and cloud metric changes.
- Applying this regression-based technique to find the most statistically relevant resource metrics from all available metrics to a specified anomaly detection requirement.
- Defining assertion specifications as predictors of the expected behavior for system operations, based on the correlation and causation model.
- Proposing a new approach to identify anomalies resulting from ripple effects of errors and distinguishing them from direct effects.
- 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.

This article is an extension of an earlier publication [16]. In addition to state-based metrics explored in our previous work, we study non-state-based metrics and leverage this exploration to address the problem of finding relevant metrics for anomaly detection. Further, we present a new approach to distinguish the ripple effect of errors from the direct effect of errors. To this end, new technical material is presented throughout sections 4, 5, and 6, respectively. In addition, we share the insights and lessons learned from this process and discuss the potential usage of the proposed approach for improving DevOps processes.

The rest of this article is organized as follows: first, we give a relevant background to this work along with the motivating example of our study in Section 2. This is followed by an overview of the proposed approach in Section 3 with its details being explained in Section 4. In Section 5, we present our experimental results from the learning phase, followed by an evaluation of error detection in Section 6. In Section 7, key related work is discussed, and finally conclusions and future work are given in Section 8.

2. BACKGROUND AND MOTIVATIONAL EXAMPLE

In this section we provide the necessary background explanation for the concept of sporadic cloud operations, discuss why cloud DevOps operations commonly fail, and

highlight some of the current challenges of operations monitoring, in particular for the domain of public cloud environments.

The focus of our work is on monitoring and dependability assurance of DevOps application operations, also referred to as "sporadic operations," in public cloud environments. Examples of sporadic operations are *Backup*, *(Rolling) Upgrade*, *Cloud migration*, *Reconfiguration*, *On-demand scaling*, *Rollback/Undo*, and *Deployment*. These operations are administrative operations that do not necessary have a scheduled routine. In other words, "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" [17]. Sporadic operations often have a system-wide impact and several technological complexities make ensuring the successful execution of these operation a challenge [18]. In the following sections we will provide further background about these challenges, followed by a case study that motivates our work.

2.1. 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 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 VM instance and its attached resources is important in its own right. From time to time an instance fails – for example, an instance freezes, crashes and/or it becomes unresponsive. Such 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 an increasd 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) [19, 20]. The occurrence of any of these failures during an upgrade operation can put the upgrade process on hold or even derail it. Hence, it is important to adopt a mechanism to track and trace the successful execution of an operation.

2.2. Concurrent Operations and Configuration Changes

Cloud sporadic operations are subject to interference from simultaneous operations, whether through automatic concurrent operations or manual changes applied to a system and its resources. 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 Auto Scaling Group (ASG), for example, 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, demonstrating that the validation of sporadic operations has critical importance.

2.3. Motivation 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 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 VM 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 an error at any time during these steps.

The contemporary practice of *continuous deployment* focuses on pushing every commit into production as soon as possible – 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 [21] to many times per day [22], updating hundreds of machines, without causing any service downtime.

In a nutshell, rolling upgrade is an excellent exemplar operation for our investigation for the following reasons: according to several reports [23], upgrade operations are

one of the most error-prone operations; the rolling upgrade operation is one of the sensitive and critical operations in Cloud system administration as a failure may cause a system-wide outage; and a rolling upgrade operation is likely to be affected by interference of concurrent operations and configuration changes, respectively. For further information about Rolling Upgrade readers can refer to [18], [24], or [25].

3. OVERVIEW OF PROPOSED APPROACH AND DATA COLLECTION

Our new approach uses a statistical technique to extract a regression-based model that explains the correlation and potential 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, for example, 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.

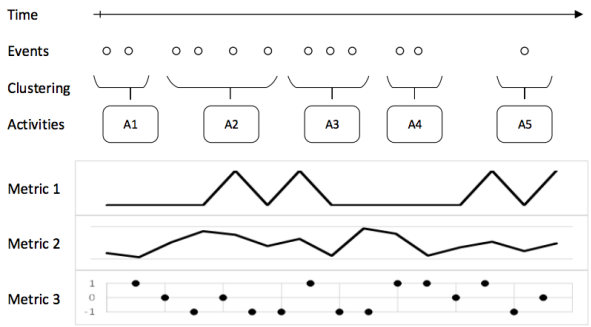


Figure 1: Assertion derivation from log and metric observations. *Note: A1..A5 are activities; Metric1..3 are metrics like CPU utilization, network usage, number of instances changes etc.

This study explores the relationship between the behavior of application operations and a clouds 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; and 5) Assertion specification for anomaly detection. We describe the steps of data collection for both monitoring metrics and logs in this section, and then we discuss in detail the steps of the approach to utilize these data for the the metric selection and anomaly detection in the next section.

3.1. 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.

3.1.1. Metrics from Monitoring Tools

Amazon AWS provides CloudWatch as a monitoring tool to track the usage and status of various resources for many Amazon services, including Amazon EC2, S3, DynamoDB, etc. In our experiments, we used CloudWatch to collect measurements for AWS resources, specifically for individual VM instances, Auto Scaling Group (ASG), and Elastic Load Balancer (ELB).

CloudWatch offers two modes of monitoring: basic monitoring and detailed monitoring. In basic monitoring mode, data is collected at five-minute intervals whereas in detailed monitoring mode, data is collected with one-minute frequency. For this study, we used detailed monitoring. Depending on the type of resources, CloudWatch may collect data at a higher frequency, but only makes it available per minute, as minimum, maximum, average, and possibly sum of the data points for each minute.[1] The data is available in the JSON file format and can be retrieved through an API.

[1]Like to AWS, the two other major public cloud service providers – at the time of writing this paper, Microsoft Azure and Rackspace – provide motoring metrics with a frequency of one minute or more.

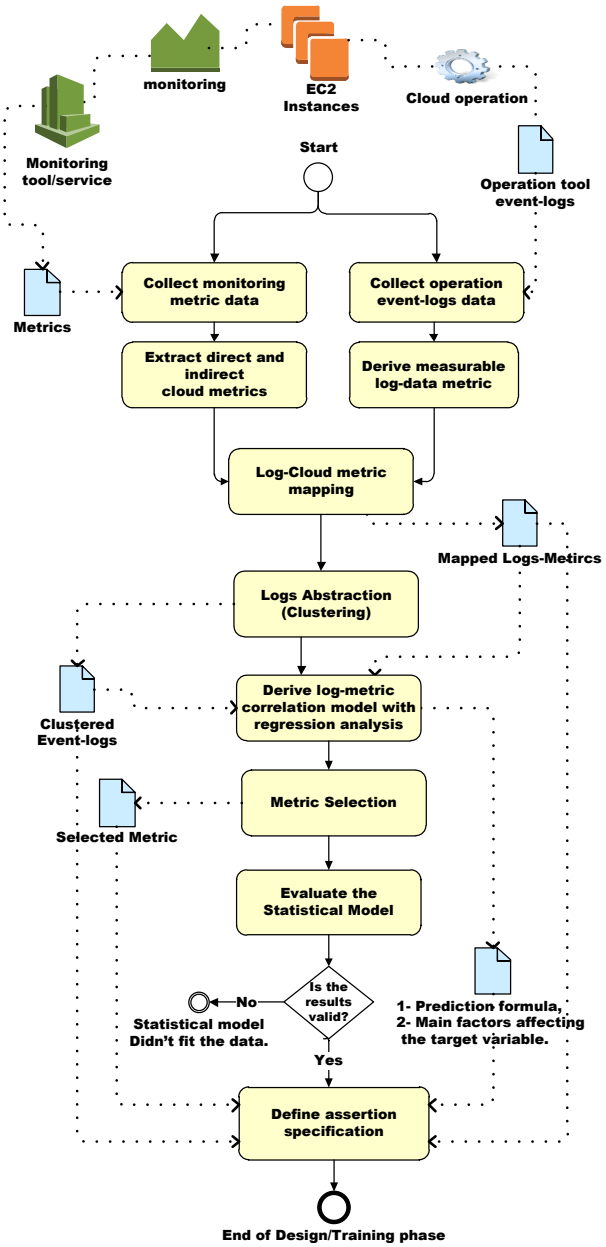


Figure 2: Workflow of extracting correlation between logs and metrics.

Monitoring tools provide multiple metrics to show the status of various system and software resources. As with CloudWatch, several monitoring metrics are available at both VM instances level (showing the status resource for each VM instance) and at group level (such as Auto Scaling Group metrics). These metrics include CPU utilization, Network traffic (both incoming and outgoing), Disk I/O (read and write), and Latency.

In general, monitoring metrics are presented in two forms. One form is related to metrics that show the percentage or the degree of the changes on the usages of the resources, such as CPU utilisation or network traffic. The other form indicates the status of a resource, for example an instance being in the "stopped" or "running" state.

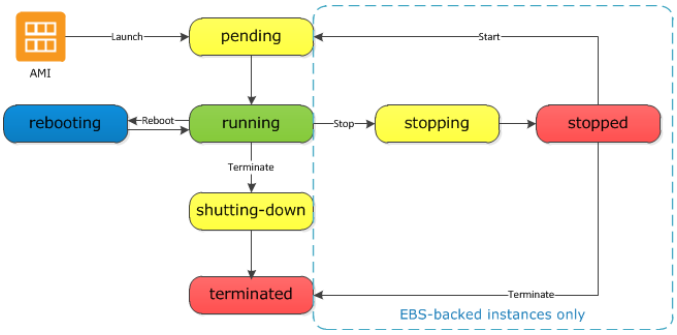


Figure 3: Amazon EC2 instance lifecycle – source: AWS documentation.

In our study on the effects 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 life-cycle of an Amazon EC2 instance.

The metrics available from CloudWatch do not explicitly show the number of started or terminated instances within an ASG. The metric "AsgInServiceInstances" shows the total number of healthy machines in an ASG, but it is only partly indicative of our desired metrics: if, during any minute, a new machine becomes active and an old one is terminated directly thereafter, 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, are only present when the machine is active. Hence, we were able to derive metrics for the numbers of started and terminated instances, respectively. These derived metrics from CloudWatch data are a cornerstone of our approach, insofar as the case of rolling upgrade is concerned.

As there is usually a large number of monitoring metrics available, a system operator needs to decide which monitoring metric or metrics to focus on. In our study, we demonstrate how to leverage a simple statistical technique to find statistically correlated metrics from all available metrics rather than merely relying on domain knowledge to separate the relevant metrics from non-relevant ones. Additionally, we investigated how effective these target metrics are for addressing our anomaly detection requirements. Lastly, we discuss how some of these metrics can become relevant to other forms of anomaly detection.

3.1.2. Event Logs from Operations Tools

To draw any mappings between cloud metrics and information from operation logs, 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.

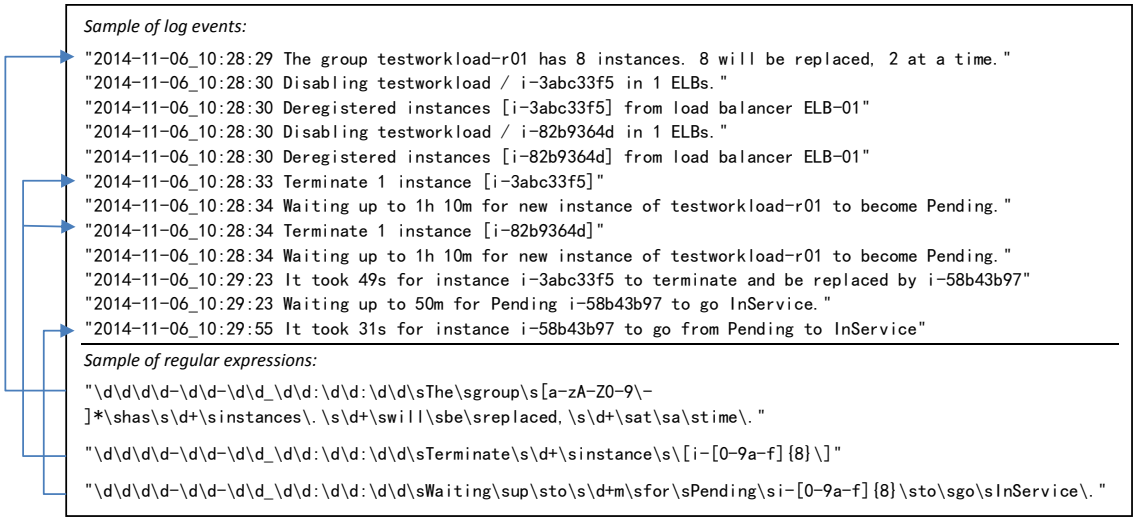


Figure 4: Sample of log event and extracted regular expression of Rolling upgrade operation with Netflix Asgard.

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*.

For our case study, we used *Netflix Asgard* both, to execute rolling upgrade operations as well as to collect operation logs. Asgard is an open-source, web-based tool published by Netflix for managing cloud-based applications and infrastructure. It automates some of the AWS cloud operations such as deployment and upgrade, and hence provides a higher-level management interface. Its log fulfils our base assumption, and contains high-quality textual messages – albeit the latter is not required in our approach. Asgard has been released publicly and is widely used.

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 and the regular expressions extracted. The regular expressions can be generated from event logs automatically as described in [26]. However, we extracted a regular expression that matches exactly the log lines belonging to each event type manually. Fig. 4 shows the sample of log events and the sample of regular expression extracted for these log events. During log processing, the event type for each log line can be identified by matching the log line to the corresponding regular expression. We can then derive a metric that shows the occurrence of each event type over time throughout the operation process. Fig. 5 shows the pattern of occurrences of the event logs throughout the process of a rolling upgrade operation for updating four virtual machine instances.

In the log occurrence pattern shown in Fig. 5, the axis that is labelled as *Logged event types* lists the event type 1 (ET01) to event type 18 (ET18); axis labelled as *Timestamp* shows the time-line of the operation; and vertical axis shows the number of occurrence of each logged event type for each minute of the operation. Looking at the figure, it reveals that there is a reoccurring pattern that is happening at different time windows. For instance, ET01, ET02, ET04, ET05, ET06, ET07 and ET08 are just appeared at the beginning of the running of the operation, which may imply to be related to the logs associated to the starting and preparation steps of the operation, while there are reoccurring pattern for event types ET09 to ET17, which may be responsible of recurring activities like VM termination. In this paper, we will explore and investigate the occurrence and correlations of these logged events in detail with statistical analysis. However, we think visualizing textual log events can be helpful for a preliminary analysis and it may be useful for better understanding of the behaviour of application operations.

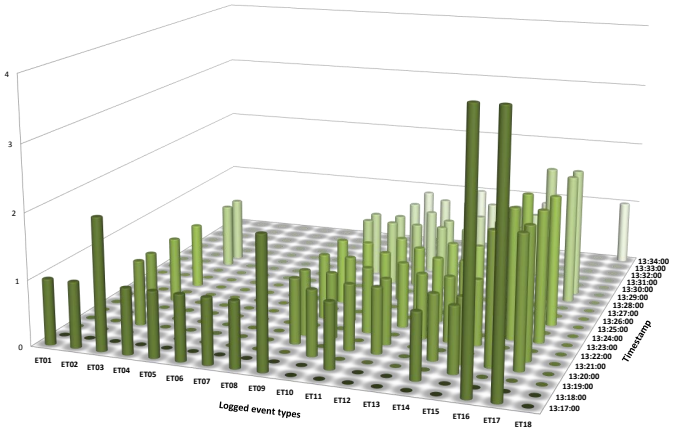


Figure 5: Visualization of occurrence of 18 event types of rolling upgrade for 4 VM instances.

4. APPROACH TO METRIC SELECTION AND ASSERTION DERIVATION FROM STATISTICAL OBSERVATIONS

In this section, we give the details of the key steps in our approach, as outlined in previous section and shown in Fig. 2.

4.1. Log-Metric Mapping

Amazon CloudWatch offers metrics with a granularity no finer than one 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, that is, only once per minute, we chose to interpolate the occurrence strength of event log types that occurred within each one-minute long time window to the respective minute. The number of occurrences for each event type is extracted as described above in Section 3.1.

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 is 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.

4.2. 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 may cause a problem of multicolinearity in some statistical models, which can 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 strong associations 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. Fig. 6 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, that is, in our case, that the events of these types (almost) always co-occur. Negative values indicate that events of the respective event types rarely co-occur. It is important to note that in this stage, we assumed that we don't have knowledge about the context of the logs and we simply leverage Pearson as a tool to find whether any correlation exist between event logs. For example, from Fig. 6 we can observe there is a strong correlation between two events of ET08, ET09, ET10 and ET11 , and perhaps these events belong to one activity. By looking at the context of the logs we can confirm that all these event types are responsible for the process of VM termination of Auto Scaling Group: ET08 and ET09 reflects the VM is going to be disabled and removed from ELB and ET10 and ET11 indicates that the VM is going to be terminated. More details of the process and results of clustering of logs to activities are explained in Section 5.1. It is important to note that clustering event logs simply based on the correlation of occurrences of the events may not lead to a set of meaningful activities. Where user interaction is a direct application, other factors should be taken into consideration, for example, as done in [26].

4.3. 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 [27]. It is important to note that while the presence of causation certainly implies correlation, the existence of correlation only implies a potential

		ET07_Group	ET08_Disabl	ET09_Remo	ET10_Termi	ET11_Waitir	ET12_ItTook	ET13_Instan
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
ET16_	Pearson Correlation	-0.073	0.126	0.125	-0.111	-0.119	-0.141	0.102
	Sig. (2-tailed)	0.098	0.004	0.004	0.011	0.007	0.001	0.021
ET17_	Pearson Correlation	-0.073	0.846	0.846	0.637	0.631	-0.25	-0.237
	Sig. (2-tailed)	0.098	0	0	0	0	0	0
	N	514	514	514	514	514	514	514
ET18_	Pearson Correlation	-0.019	-0.073	-0.073	-0.074	-0.074	-0.064	-0.061
	Sig. (2-tailed)	0.672	0.097	0.097	0.094	0.092	0.146	0.169
	N	514	514	514	514	514	514	514

Figure 6: Snippet of a Pearson correlation matrix .

causation. For instance, one may observe a correlation between power consumption and the number of errors. However, the underlying cause of a higher number of errors could be due to the increased chance of observing any error when a higher number of VMs is involved as the result of the 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. In order to infer whether a correlation implies causality, one needs to ensure that the correlation is extracted from a controlled environment, that is, to ensure 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 in finding the effect of operation actions on cloud resource state changes. To this end, we start from data that has been collected over a period of time and has resulted in a sufficiently large number of data points. We then use a regression-based technique to discover correlation between logged events and changes in metrics, that is, the absence, presence, and strength of such changes. In our running example, whenever there is a log event indicating that a termination request for one VM has been issued, the expectation is that within the next minute, one VM will transition from "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" [28]. 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 [29]. 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 (i.e. event logs), and one dependent variable (DV) (i.e. a monitoring metric) as the outcome.

Given y is the dependent variable, $x_1, x_2, \ldots, x_n$ the independent variables, and ϵ to be the error term or noise, the general form of a linear regression function is

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon \quad (1)$$

where the intercept α denotes a constant value that is the expected mean value of y when all $x = 0$. The coefficients $\beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$ 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. We analysed our data with multiple regression and generalized linear regression models, including *Poisson regression* and *Negative Binomial regression*, respectively. It is beyond the scope of this paper to explain in detail the results 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 (for example 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.

4.4. Target Metric Selection

In cloud resource monitoring, there are usually many monitoring metrics available. For example, AWS CloudWatch offers 168 metrics for an ASG with 10 machines, an ELB and an Elastic Block Storage (EBS).[2] When administrating tens or hundreds of machines, tracking changes on each metric is often impractical and, in many cases, not immediately beneficial. In fact, system operators are often exposed to an excessive amount of monitoring information, and they receive too many monitoring warnings and alerts [12, 13, 14]. Other problems include a system operator receiving too many false alarms or a flood of alerts from different channels about the same event [30].

This excessive information load can make the detection of anomalies more complicated, and it may delay the detection of an issue at critical times. Therefore, it is important to identify which subset of monitoring metrics is most relevant for a specific monitoring requirement. In today's practice of system monitoring, this task is done by system operators mostly manually, based on their domain knowledge. In fact, most of software monitoring packages offer customization, often allowing the choice which metrics to show and user-defined alarms. However, this customization has become more difficult with the ever-growing number of resource types in cloud environments and the variety of associated metrics.

[2] At the time of writing this paper, Amazon AWS provides metrics monitoring for 30 AWS services including 14 metrics per EC2 instance, 8 metrics per ASG, 10 metrics per ELB and 10 metrics per EBS. For further information refer to: http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/CW_Support_For_AWS.html

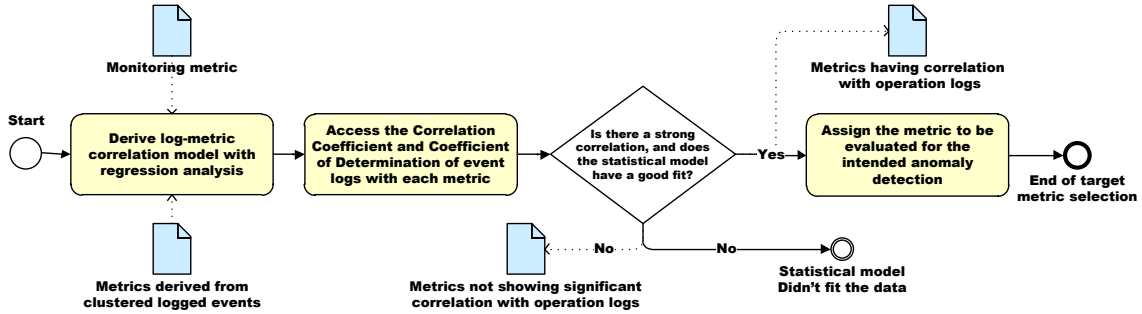


Figure 7: Checking the relevancy of a monitoring metrics.

We developed a new approach that facilitates the process of metric selection, primarily by statistical analysis and objective insights, i.e., the statistically most relevant target metrics. In our concrete use case, we investigate all the group-level metrics like CPU utilisation and Network input/output, as well as state-based metrics like StartedInstances and TerminatedInstances using regression analysis as explained in Section 4.3. Once a multiple regression equation has been constructed, we can check how strong the regression output is in terms of (i) correlation of the event logs with the target metrics, and (ii) the model's the predictive abilities.

The correlation is assessed by extracting the Correlation Coefficient, denoted by R, and the predictive ability is examined by the Coefficient of Determination, denoted by R^2 (R-squared). The values of both R and R^2 lie between 0 and 1. R measures the strength and the direction of a linear relationship between independent variables (log events) and the target variable (a monitoring metric). In contrast, R^2 shows the proportion of variance in a target variable (y) that is explained by predictors (x_1, x_2, $x_i, ...x_n$). This allows us to determine how certain one can be in predictions derived from a given model. The closer R^2 is to 1, the better the model and its prediction. Therefore, we use the coefficient of determination R^2 to identify whether the behaviours of metrics are significantly influenced by the operation's activities. Otherwise, the metric will be identified as potentially not being influenced by the operation's activities. Fig. 7 illustrates these steps. It is important to note that a high value for R^2 indicates a metric has *linear* relationship with the activity logs of an operation, and such a metric can be *potentially* employed for the anomaly detection of an operation. When a metric does not show a correlation with the operation's activities, either there is no direct relationship between them or there might be a non-linear relationship that could be explored further with non-linear regressions.

4.5. Assertion Derivation for Fault Detection

Once the correlation and causation relationships between events and metrics is determined, these can be formulated as assertions, such that an assertion evaluation service can determine at run-time if the assertions are fulfilled. If any assertion is violated, the service will raise an alarm, for example, 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, x_2 and x_3 refer to the relevant activities. The resulting regression equation is

$$y = \alpha + \beta_1 * x_1 + 0 * x_2 + \beta_3 * x_3 + \ldots + 0 \quad (2)$$

From the model, we learn concrete values for α and the β_i. In particular, for any x_i where the explanatory analysis of a correlation is not statistically significant, where ($p > .005$) or Standardized Coefficient is close to zero, we set $\beta_i = 0$. At run-time, 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, x_2 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 and terminated VMs), the prediction always has a *continuous* result value. This value then needs to be discretized. The easiest method for discretisation is rounding the number, but this may lead to predictions with a fairly low confidence. For example, how do we interpret the meaning of *half* a VM was terminated when $y=0.5$? Another method is to define a threshold t, such that $0 < t < 0.5$, and the prediction is set to the natural number i closest to y *iff* y is closer to i than t, that is, $|y - i| < t$. Finding a suitable threshold t has to be done for each application scenario separately, as a trade-off is needed between missing too many real alarms (false negatives) and receiving too many false alarms (false positives).

5. EXPERIMENTAL RESULTS FOR LEARNING

In this section, we discuss the experiments conducted for the learning phase – the next section covers the evaluation of error detection. We performed our analysis based

on two separate case studies of rolling upgrades: first, running multiple runs of rolling upgrade of 8 instances, upgrading two instances at a time; and the other, running rolling upgrade of 40 instances, upgrading four instances at a time. We conducted 10 rounds for each case of running rolling upgrade on Amazon EC2, and gathered 283 and 514 logged events respectively for 8 and 40 instances, as well as monitoring data from CloudWatch.

5.1. 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 when they had correlation strength of more than 75% (Pearson-r $>$ 0.75) whereas values show highly statistical significance (i.e. *p-value* < 0.01). In other words, as a rule, any event type of an activity should indicate at least 75% correlation with any other event type of the group that formed an activity. One may choose a higher or lower level depending 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 differences in correlation values, the log abstraction led to identical clusters. 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 evaluate how meaningful our log abstraction result is, we investigated the context of the log entries; the result, given in Table 1, shows that all the event types of each cluster are meaningfully related to each other. For instance, the four events *DisablingXInELB, RemoveInstanceFromELB, TerminateInstance*, and *WaitingInstancesPending* that are clustered 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. The full logs to activity mapping is given in Table 1.

We compared our result of creating a log abstraction with the one reported in a previous study [10] where the same operation (i.e. rolling upgrage) was extracted based on domain knowledge expertise. In the comparison, we did not find any conflict in terms of mapping event logs to activities, although the level of abstraction is slightly different in [10]. Based on the resulting similarity, we concluded that the derived log abstraction was meaningful and appropriate for further analysis with regression.

5.2. Identification of Relevant Target Metrics

To investigate the relationship between the occurrence of event types and cloud metrics in our regression model, we assigned the activities derived from log clustering as *predictor variables* and monitoring metrics from Cloudwatch as *candidate target variables*. We aimed to identify which metrics have the highest potential to reflect the effect of running rolling upgrades in the system.

Several metrics are available in CloudWatch, both at the instance level and at the group level. Considering instance-level metrics is, in general, impractical, especially if hundreds of VM instances are to be considered. Further, metrics associated with instances cease to exist once the corresponding VM goes out of service. Therefore, we consider group-level metrics of instances and metrics of the EC2 Auto Scaling Group (ASG) and the Elastic Load Balancer (ELB). To this end, we managed to obtain data for 17 group-level metrics based on above-mentioned experiments.

We performed multiple regression analysis to predict the value of monitoring metrics given the six activities extracted from logged events, from the 20 total runs of rolling upgrade for 8 and 40 instances. The results of linear regression analysis over group-level monitoring metrics are shown in Table 2.

In the table, R denotes the correlation between a given monitoring metric and occurrences of activities from the event logs, and R^2 indicates how well the model predicts new observations. In general, R^2 is used to assess the predictive power of the regression model for the given predictors and target variables. $Adj.R^2$ is a modification version of R^2 that adjusts for the number of predictors in a model. The value of R^2 may increase by chance if new predictors are added to the model, leading to over-fitting of the model. $Adj.R^2$ takes this into account by penalizing models with more variables, meaning that the increase in

Table 1: Event logs to activity abstraction for the rolling upgrade operation

Event	Activity
ET01_StartedThread ET02_UpdatingLaunchWithAmi ET03_CreateLaunchConfig ET04_UpdatingGroupXToUseLaunchConfig ET05_UpdateASG ET06_SortedInstances ET07_XInstancesWillBeReplacedXAtATime	A1_Start of Rolling upgrade (sorted instances)
ET08_DisablingXInELB ET09_RemoveInstanceFromELB ET10_TerminateInstance ET11_WaitingInstancesPending	A2_Remove Instance from ELB and Terminate Instance
ET12_ItTookXminInstanceToBeReplaced ET13_InstanceInLifeCycleStatePending ET14_WaitingForInstanceToGoInService	A3_Instance Replacement Process
ET15_ItTookXminInstanceToGoInService ET16_WaitingForInstanceToBeReady	A4_New Instance to go in service
ET17_InstanceXIsReady	A5_Instance is ready
ET18_Completed	A6_Rolling upgrade completed

Table 2: Coefficient Correlation and Coefficient Determination results for each metric

	Experiment: 8 Instances				Experiment: 40 Instances			
Metric	R	R^2	$Adj.R^2$	$p\text{-}value$	R	R^2	$Adj.R^2$	$p\text{-}value$
CPUUtilizationAverage	**0.751**	0.564	0.555	0.000	**0.801**	0.642	0.638	0.000
CPUUtilizationMinimum	0.391	0.153	0.134	0.000	0.367	0.135	0.124	0.000
CPUUtilizationMaximum	**0.810**	0.656	0.649	0.000	**0.855**	0.732	0.728	0.000
NetworkInAverage	0.532	0.283	0.267	0.000	0.299	0.089	0.079	0.000
NetworkInMinimum	0.428	0.183	0.165	0.000	0.640	0.410	0.403	0.000
NetworkInMaximum	0.214	0.046	0.025	0.420	0.166	0.027	0.016	0.028
NetworkOutAverage	0.502	0.252	0.236	0.000	0.299	0.090	0.079	0.000
NetworkOutMinimum	0.475	0.226	0.209	0.000	0.635	0.403	0.396	0.000
NetworkOutMaximum	0.349	0.122	0.103	0.000	0.118	0.014	0.002	0.313
InServiceInstances	0.771	0.595	0.586	0.000	0.676	0.457	0.450	0.000
ELBLatencySum	0.405	0.164	0.146	0.000	0.118	0.014	0.002	0.304
ELBLatencyAverage	0.430	0.185	0.167	0.000	0.060	0.004	0.008	0.934
ELBLatencyMinimum	0.169	0.029	0.007	0.235	0.052	0.003	0.009	0.968
ELBLatencyMaximum	0.458	0.209	0.192	0.000	0.125	0.016	0.004	0.239
ELBRequestCount	0.091	0.008	0.000	0.808	0.135	0.018	0.007	0.152
StartedInstances	**0.828**	0.686	0.679	0.000	**0.884**	0.782	0.780	0.000
TerminatedInstances	**0.921**	0.848	0.845	0.000	**0.955**	0.912	0.911	0.000

R^2 (i.e. the improvement of the fitting) must be reasonably large for the inclusion of a new variable to cause an increase in $Adj.R^2$. Thus, in our analysis, $Adj.R^2$ was the most appropriate measure to assess the relevance of each of monitoring metric and the activities of the rolling upgrade operations. Prediction abilities of the metrics based on the value of $Adj.R^2$ are shown in Fig. 8.

For several metrics, highlighted in bold in Table 2, we observe high values of R^2 and $Adj.R^2$ which suggests that the variation of these monitoring metrics can be explained by our regression model. In other words, given operation logs the model is capable of predicting the value of monitoring metrics for a few of the metrics to a fairly good extent. However, for the rest of monitoring metrics, the regression model did not fit the data, and a derived regression equation may lead to weak predictions. As shown in Fig. 8, the best fit of the model is obtained for *TerminatedInstances* in the experiment with 40 instances: 91.2% of the variation in *TerminatedInstances* can be explained by the linear relationship between six predictor variables derived from event logs and the *TerminatedInstances*. In this case, $R^2 = 0.912$ and $Adj.R^2 = 0.912$ are statistically significant at confidence level $p < 0.005$. A similar interpretation of the model can be inferred from Table 2 for the experiments with 8 instances and 40 instances for other metrics.

The results show that seven metrics have a *p-value* greater than .005, indicating that our linear regression analysis does not show a good fit for these metrics. Therefore, they are not valid candidates for our anomaly detection requirement. These seven metrics include all five ELB metrics. Other metrics such as *NetworkInput* show relative correlation with the operation's event logs. However, correlation and prediction power are not strong enough to be considered as potential candidates for our anomaly detection. Based on the above results, we select the four metrics that had the best prediction precision for further analysis: *TerminatedInstance*, *StartedInstances*, *CPUUtilizationMaximum* and *CPUUtilizationAverage*.

5.3. Correlation and Causality Learning with Multiple Regression Model

In the previous section, we identified metrics that had a strong correlation with the activities of the rolling upgrade operation. We now need to find out which of these metrics are useful for the purpose of our anomaly detection requirements. We articulate our anomaly detection requirement as follows:

To verify the successful execution of upgrading VM instances to a new version using the rolling upgrade operation in the environment of Amazon EC2.

In other words, we aim to detect anomalies during an operation's execution as mismatching values in the comparison of the actual value of a target metric with a predicted value, which we calculate from a regression equation. To this end, we explore the correlation and causalities of our regression model to assess how well we can leverage it to satisfy the above anomaly detection requirement.

One of the objectives of performing multiple regression analysis was to find an explanatory relationship between independent variables (activities) and the dependent variables (cloud metrics). In Section 5.2 we found the metrics that have correlation with overall operation's activities, yet we need to find out which of *operations' activities* are affecting a target metric to derive an assertion specifications for our anomaly detection. We thus seek to distinguish the activities of the cloud operation that are likely to affect one of the target metrics from the others. In order to perform such analyses, we considered the Correlation Co-

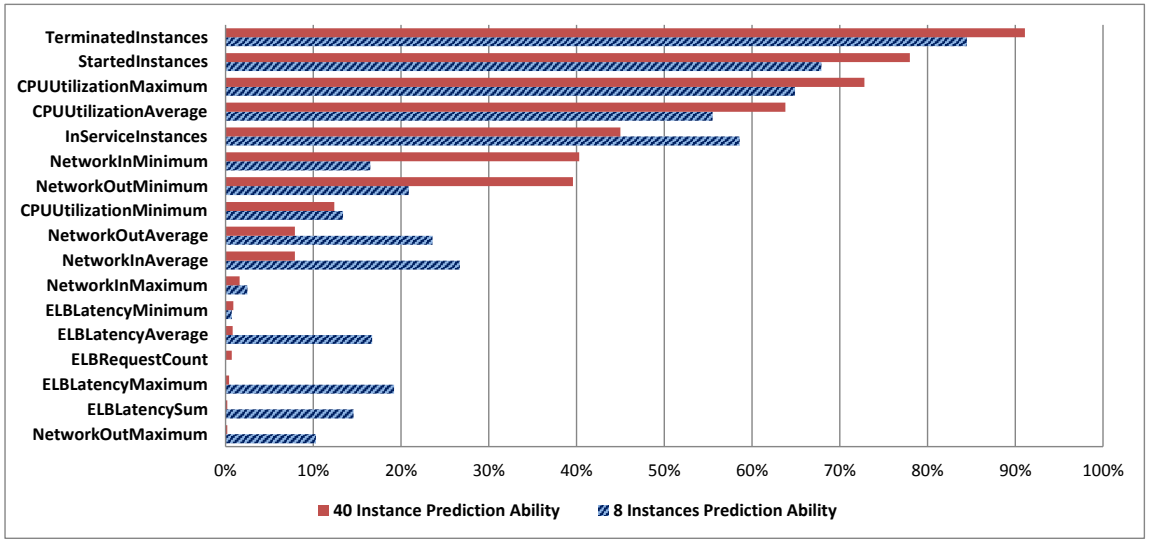


Figure 8: Prediction ability for each monitoring metric based on $Adj.R^2$.

efficient results for each predictive metric of our multiple regression models generated by regression analysis.

We will first explain the process we used by applying it to the case of the *Terminated-Instances* metric and the experiments with 40 instances. This is followed by a summary of the results for the *StartedInstances* and *CPUUtilization* metrics, respectively.

Table 3 shows the Coefficient Correlation resulting from the generated regression analysis. By considering the *p-value*s in Table 3, we observe that the correlation of the metric and activities A3 and A4 are statistically insignificant ($p > .005$). Therefore, we conclude that the corresponding two variables cannot explain the variation of the target variable. Further, the standardized regression coefficient shows that the contribution of activities A1 and A6 are almost zero, and can also be excluded. These observations allowed us to narrow the set of contributing activities down to A2 and A5. Rerunning multiple regression with only these two activities resulted in the outcomes shown in Table 4.

Given the high difference between the Activity A2 (0.836) and the Activity A5 (0.15), presented in Table 4, it can be concluded the Activity A2 is the main cause of termination of an instance. The context of the activity log confirms that the model was effective in correctly identifying the activities that causes the termination. These findings can then be used to define an assertion specification, as explained in Section 4.5, with the following equation:

$$y = 0.051 + 1.197 * A2 + 0.149 * A5 \quad (3)$$

We applied a similar approach for the regression analysis of other candidate metrics. Fig. 9 shows the importance or relative contribution of each predictor for each candidate metric, similar to the one that is explained for the TerminatedInstances in Table 3, Table 4 and Equation 3.

Our key anomaly detection objective articulated at the beginning of this section was: which of these metrics is most suitable to be leveraged for verification of successful execution of a rolling upgrade operation? To find the answer to this question, two factors need to be considered: (i) which metrics have the best prediction ability using statistical information, and (ii) what types of metrics are best suited for the purpose of anomaly detection.

In regards to the first consideration, among the available metrics few show a high correlation in the regression model. The predictive ability of *TerminatedInstances* is the highest with $R^2 = 0.912$ and $Adj.R^2 = 0.911$. Such factual information can be a strong aid for an operator when filtering out all but the most relevant metrics, and to understand which metrics are affected by which activities in an operation.

In regards to the second consideration, given the anomaly detection requirement for a rolling upgrade operation, state-based metrics that would reflect the target changes of the rolling upgrade operation should be the first candidates. This is important because our anomaly detection has the objective to find unintended changes of number of terminated and updated VMs while the upgrade operation is running. Therefore, looking at state-based metrics is a rational choice, and thus we considered *TerminatedInstances* and *StartedInstances* as the best suited metrics for anomaly detection. Nevertheless, for the sake of comparison between the results from an state-based metric (e.g. TerminatedInstances) and non-state based metrics (e.g. CPUUtilization) we demonstrate the evaluation result of both type of metrics in the next section.

Note that other (types of) metrics might suit anomaly detection with different goals better. For example, if we had the objective of detecting performance anomalies of our cloud resources during rolling upgrade operations, then a CPU utilization metric could be hypothesized to be a

Table 3: Coefficient Correlation - 40 instances - Terminated-Instances Metric

Predictors	β	Std. Error	B	*p-value*
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	0.150	0.000
A6_Rolling upgrade completed	-0.730	0.184	-0.058	0.000

*Note. β = Unstandardized regression coefficient;
B = Standardized regression coefficient.

Table 4: Coefficient Correlation for identified influential factors - 40 instances - Terminated-Instances Metric

Predictors	β	Std. Error	B	*p-value*
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.

more appropriate metric. Such outcomes could also be employed for dynamic reconfiguration of cloud auto-scaling policies when a rolling upgrade operation is running, to effectively counter-balance the impact of running these types of sporadic operations.

A question a reader may asks is how much is needed to relearn when the conditions and configuration of systems change. In our case study, the learning process has been conducted from two different learning data sets and both almost lead to the same conclusion, though the experiment with 40 instances provided slightly more accurate learning as there were more records of data to be utilized for our statistical analysis. As far as we have sufficient data for a statistical analysis, one can be sure that even in the case of changing conditions the result of identifying selected metrics and the log event predictors to affect a particular metric will still be the same. For instance, the metric of TerminatedInstances will always show high correlation with activities of rolling upgrade (Fig. 8), or A2 in logs (Fig. 9) will always have the highest impact on changing the TerminatedInstance metric in compare to other activities. However, how much the impact of activities on metrics will change if the system condition or configuration changes needs further investigation.

6. ANOMALY DETECTION EVALUATION

In this section, we describe how we applied our approach to error detectiong in the rolling upgrade case study. To this end, we injected faults into 22 runs of rolling upgrade, and used our learned model for prediction and fault detection. Additionally, we address the cases of anomalies that result from ripple effects of faults, and present our technique that can distinguish them from direct effects of faults. Key insights and lessons learned from our experiments are discussed at the end of the section.

6.1. Evaluation Method

In order to evaluate how well the derived assertions can detect errors, we conducted a second experiment which was run independantly from the one used to learn the model. The raw data of this experiment was obtained from experiments run by our colleagues [14]. The experiments were conducted on Amazon EC2, upgrading 8 instances, two 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 5,335 lines of logs emitted by Asgard. A total of 115 faults were injected at random.

As explained in Section 5, the equations derived from multiple regression models can be used to predict the number of started and/or 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 raised. We wanted to find out how accurately our approach could identify anomalies. In any given time window, in case there are concurrent anomalies happening in the runtime, our approach could identify most of these anomalies. These anomalies, such as sudden termination of a VM or a peak on CPU usage, are distinct from the impact of a rolling upgrade operation in our approach because the effect of the activity of rolling upgrade operations on the status of resources has already been taken into account to calculate the predicted value.

Since the injected faults were all VM failures, our approach tries to distinguish between VMs being terminated due to legitimate operational activity, and termination caused by fault injection. We chose to inject faults that caused VM failures 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 anoma-

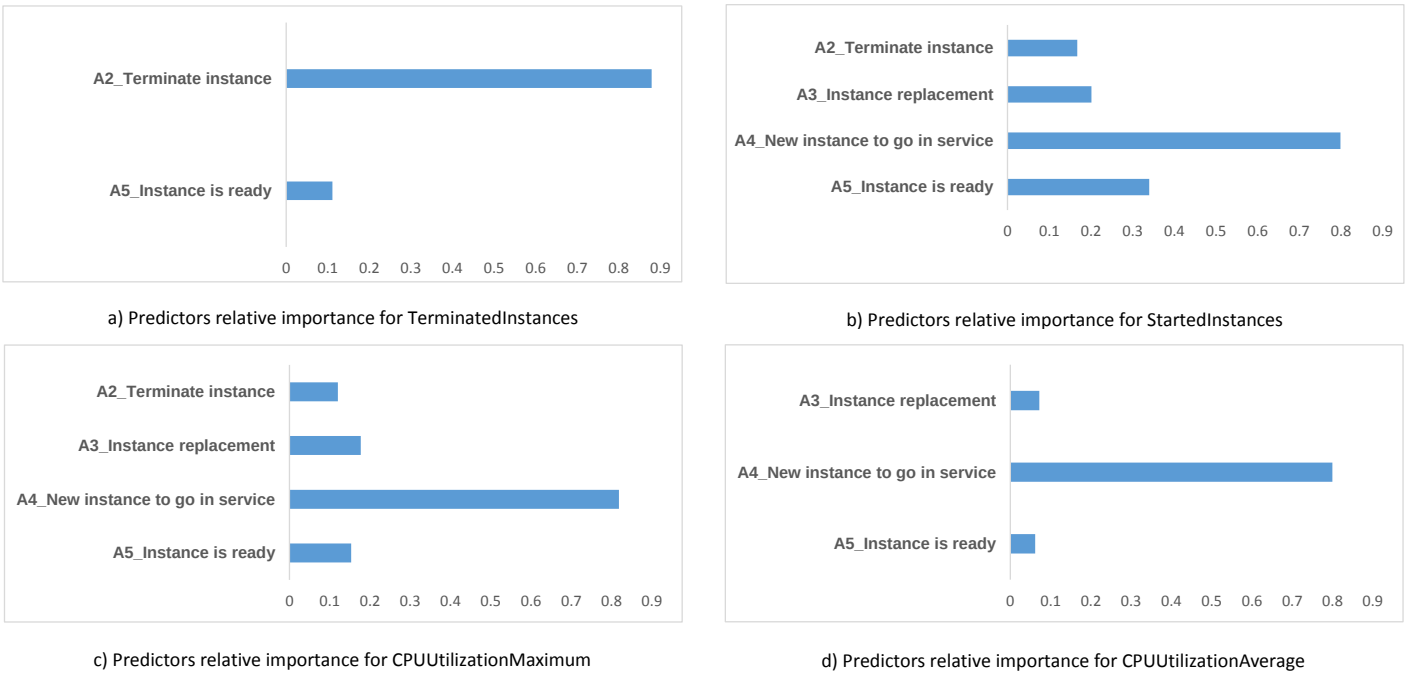


Figure 9: Predictors relative importance for selected monitoring metrics based on Standardized Coefficient(B)- bigger value indicates higher contribution of an activity to the changes in a metric.

lies. Therefore, it was reasonable to inject fault types that cause VM termination rather than other types of faults.

The faults were injected automatically to the VMs every three to 6 minutes by a software service that was running in parallel with the rolling upgrade operation. In our experiments each fault has been injected into one VMs separately. There have been cases where a maximum of two VMs went out of service due to two separate fault injections during one time window. It is worth mentioning that rolling upgrade operations can be a time consuming process and they may target tens or hundreds of VMs, it is possible that more than one failure can occur during one rolling upgrade operation and thus we thought having more than one fault being injected in rolling upgrade operation would be more realistic and thorough evaluation. When an anomaly is detected during a time window an alarm is issued containing the information of difference between the expected value calculated from regression equation based on log events versus the actual number of termination occurred in each minute as indicated in metric.

In order 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 5 explains these four categories in terms of an alarm being raised (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 the 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: $P = \frac{TP}{TP+FP}$. Recall is a measure of completeness of correct alarms: the percentage of injected faults where an alarm was raised: $R = \frac{TP}{TP+FN}$. The F-measure (or F_1-score) is the weighted average (harmonic mean) of precision and recall: $F_1 = 2 * \frac{P*R}{P+R}$.

6.2. Evaluation Result with State-Based Metric

In our study, we observed possible delays between an operation action and its effect(s) becoming observable. For instance, consider the duration of terminating one VM: the time between the respective event being logged and the VM actually being terminated may vary between 15 seconds and 3 minutes. It is thus not uncommon that a VM is terminated in one minute, but the CloudWatch metrics only reflect the termination in the next minute, or possibly 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 for prediction: zero minutes (0mTW), that is, only the current minute; one minute difference (1mTW), that is,

Table 5: Classification metric for the generated alarm

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

Table 6: Evaluation results of state-based metric – basic detection.

Evaluation Metrics	0mTW	1mTW	2mTW
Precision	0.567	0.712	0.745
Recall	0.670	0.914	0.921
F_1-Score	0.706	0.826	0.849

the current minute, the minute before, and the minute after; and two minutes difference (2mTW), that is, from two minutes before to two minutes after. It should be noted that a longer time window 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?

Ripple effects may occur when the model predicts a particular change to one (or more) metric values, but due to the fact that an anomaly has already occurred (e.g., a VM instance prematurely terminated), the predicted change does not occur. This can lead to further false alarms at a later stage of the operation's process.

The results of monitoring the operation based on the metric of TerminatedInstances and log context with the three different time windows are shown in Table 6. Precision, Recall, and F_1-score are given without considering the impact of the ripple effect of injected faults. The reader may note that there is a significant difference between the basic precision value of 0mTW and 1mTW: 0.145 (or 14.5%). The difference between 1mTW and 2mTW, in contrast, is comaratively smaller. 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, respectively.

Not all the effects of injected faults result in observable errors immediately. There are cases where errors have ripple effects. Table 7 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 6). To distinguish actual failed prediction from ripple effects (where the prediction behaved as expected), we analyzed all 30 FP and 7 FN

Table 7: 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 pending to be started.

Table 8: Evaluation results with state-based metric – detection result with ripple effect

Evaluation Metrics	1mTW_Ripple Effect	2mTW_Ripple Effect
Precision	0.923	0.925
Recall	1.000	1.000
F_1-Score	0.960	0.961

cases for 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 8 and in Fig. 10 respectively.

6.3. Evaluation Result with Non-State Based Metrics

In the previous section we demonstrated and disscussed the anomaly detection performance utilizing TerminatedInstance as the main monitoring metrics. In our metric selection process in Section 5.2 we observed non-state based metrics of CPUUtilizaionMaximum and CPUUtilizationAverage also have fairly good correlation with the rolling upgrade log activities, though not as high correlation as the TerminatedInstances. In this section we follow the same approach that was explained in detail in the previous section and show the result of anomaly detection when using CPU-related metrics and compare this with the results we obtained when using the TerminatedInstancs metric.

In the approach for the state-based metric the threshold value is an indicator of whether there is a change of state occurred or not, and whether that matches with the normal behaviour of the system, as with non-state based metrics such as CPU-Utilization a threshold indicates the value that separate the outliers from the range of normal values of the metric. In most statistical based anomaly detection techniques, Standard Deviations (σ) from Mean are used to detect outliers, often the values dispersed above $\pm 2.5\sigma$ to $\pm 3.0\sigma$ are considered outliers [31]. In our approach, the metric threshold indicates the acceptable range of difference between the values calculated from the regression formula and the actual value of the CPU utilization. Any observed value above these range are considered anomalies. In our experiment, we consider the difference of predicted value versus actual value of CPU utilization that are within one Standard Deviation $\pm 1\sigma$ from Mean to be normal, otherwise an alarm is registered.

Similar to the case of TerminatedInstances, we have used two separate datasets for learning and evaluation. In both learning and evaluation systems, the monitored system was exposed to a workload with average of 40% of EC2 Auto Scaling Group aggregated CPU power with Variance of 4.2 and Standard Deviation of 2.05. The systems under test for evaluation were also exposed to an additional CPU workload task of average 20% of EC2 Auto Scaling Group aggregated CPU power that lasted between two to three minutes, and with injection frequency of every two minutes.

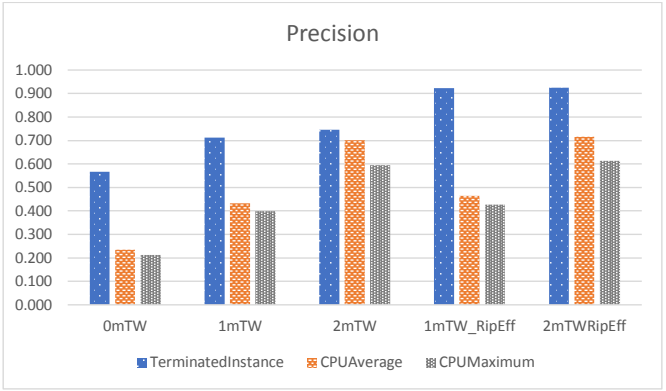
In contrast to general anomaly detection where that metric is the main source of information, we have contextual information for the operation and we obtained the approximate effects of operation activities on resource consumption from the learning dataset through the regression analysis. Among all the log activities of the operation, Activity A4 (New instance to go in service) had the highest impact on CPU utilization, Fig. 9 while A2, A3 and A5 had very low impact on CPU Utilization and A1 and A6 did not show an observable impact. We took into account the impact of these activities based on Equation 3 and calculated the Precision, Recall and F-Score. The result is shown in Table 9 and Fig. 10.

As shown in the Table 9 and in the figures, the overall results of both precision and recall for a zero minute time window is very low for the both CPUAverage and CPUMaximum metrics, then with expanding time window to one minute before and after the collected data point the result shows improvements. Finally the best results were obtained by using two minute time windows. There are two reasons that we observed a delay of the effect of failure of VMs with CPU consumption: one is similar to the effect for TerminatedInstances, as the action of termination of instances varies between 30 second to over two minutes; second reason is the activity associated to process of an instance to go into service. The activity A4 has the highest CPU consumption according to our findings demonstrated in Fig. 9, and it takes place right after the termination process of the instance is completed.

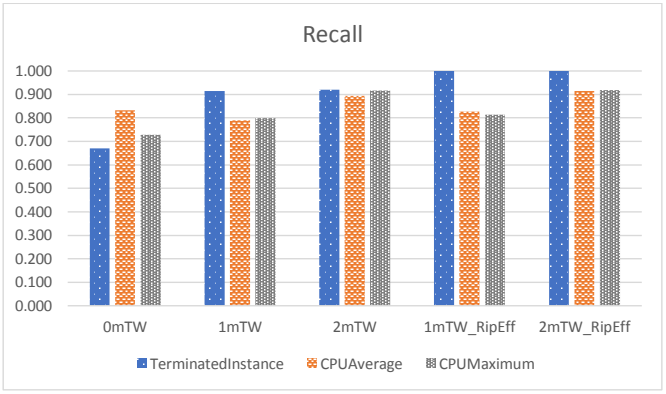
The reader may note that there is a considerable gap between anomaly detection delay between TerminatedInstance and CPUUtilization. Such an observation can be explained as follows: for the case of detecting anomalies with TerminatedInstance, we had the information to decide when the termination happened and whether it was the result of a legitimate process or our fault injection, and that helped us to detect failures as soon they occurred. While for detecting failure with metric-based on CPU utilization, the significant symptoms are observed after termination, when a new machine is under the process of going into service.

6.4. Ripple Effect Detection

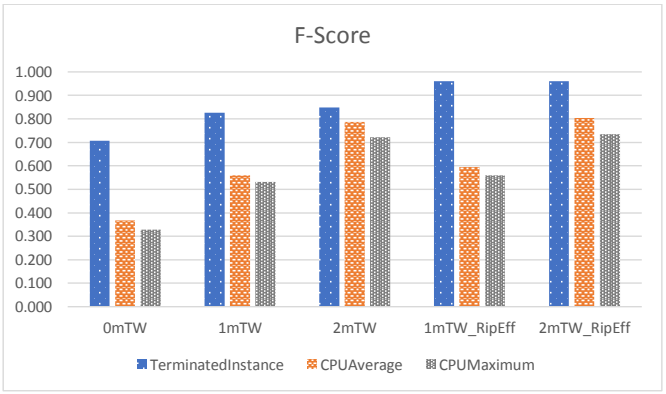
In the previous section we briefly discussed the presence of ripple effects of errors and the way our approach identified them as anomalies. We demonstrated that our



a) Precision



b) Recall



c) F-Score

Figure 10: Results for three different time windows and with ripple effects for CPUAverage and CPUMaximum vs. TerminatedInstance.

proposed approach was effective in detecting injected failures with a high precision. Nevertheless, we noticed that a small portion of reported anomalies were not caused by the direct effect of errors. After a detailed analysis we found that the vast majority of these anomalies were resulted from ripple effects of errors that had already been detected.

Anomalies detected in a system are reported through

Table 9: Evaluation Result with Non-State Based Metrics

	CPUUtilizationAverage					CPUUtilizationMaximum				
Metric	0mTW	1mTW	2mTW	1mTWRipEff	2mTWRipEff	0mTW	1mTW	2mTW	1mTWRipEff	2mTWRipEff
Precision	0.235	0.433	0.701	0.464	0.716	0.212	0.399	0.595	0.427	0.614
Recall	0.833	0.789	0.895	0.828	0.914	0.728	0.798	0.917	0.814	0.918
F-Score	0.367	0.559	0.786	0.594	0.803	0.329	0.532	0.721	0.560	0.736

some form of alerts or notifications. As discussed previously, excessive amounts of less important alerts and notifications can cause alert fatigue, and this concern also applies to the detection of anomalies caused by ripple effects. Therefore, our concern in this section is how to distinguish if a detected anomaly stems from a direct error or from the ripple effect of an error, with the goal to suppress alerts from the latter.

In order to address this issue, we implemented a mechanism to automatically detect ripple effects of errors. To do so, we kept track of the instance identifiers, which are present in both the metrics data and in the operation's logs. Additionally, we already had the timestamp of the event logs – the time that the termination of instances has been triggered. Given this information, as well as the records of detected anomalies, we determine if a raised anomaly is related to an already affected VM or not; if not, we tag the anomaly as a ripple effect and suppress the respective alert. It worth mentioning that although the mechanism of ripple effect detection here is presented separately, this is a part of the integrated process of anomaly detection. In fact, in the whole experimental method, any detected anomalies are checked if it is caused by an actual fault injection or whether it is a ripple effect symptom of an injected fault. In other words, the ripple detection is applied to the whole experiment of fault injection; for all the anomalies reported there is an action, if it is direct failure there will be an *error alert*. If it is a ripple effect then there will be a *warning alert*. The pseudo-code of this ripple effect detection algorithm is shown as Algorithm 1.

We observed three types of ripple effects in our experiments, as shown in Table 7. Out of the total of 28 ripple effects, five were related to our fault injection process and thus they had no effect on the rolling upgrade process. Out of the remaining 23 ripple effects, by applying Algorithm 1 our approach managed to automatically detect 21. These ripple effects were related to the VM instances that the rolling upgrade operation intended to terminate and replace, but the instance had already gone out of service due to injected faults. The remaining two ripple effects could not be detected automatically; they were associated with two VM instances that were in the state "pending to be started" when the faults were injected, hence fault injection did not cause the VM instances to be terminated.

Algorithm 1 Ripple effect detection in the process of anomaly detection

```
1: while rolling upgrade is not completed do
   - Input:
2:     opsLogs ← Read Logs at each minute
3:     metricsActual ← Read metric at each minute
   - Anomaly Detection:
4:     metricEstimated ← Estimate metric with regression equation
5:     if metricEstimated = metricActual then
6:         anomaly ← false
7:     else
8:         anomaly ← true
9:     end if
   - Ripple Effect Detection:
10:    if anomaly = true then
11:        failedVMs ← retrieve latest list of VM instances that were
   failed
12:        if the error has been already reported then
13:            anomalyType ← rippleEffectWarning
14:        else
15:            anomalyType ← directErrorAlert
16:        end if
17:        ReportAnomalyWithAnomalyType(anomalyType)
18:    end if
19: end while
```

6.5. Insights and Lessons Learned from the Integration of Log-Metric Monitoring

We have shown that our approach is effective in detecting anomalies whilst cloud rolling upgrade application operations were running. It is worth noting that the proposed method is a non-intrusive approach: it does not require changes to cloud application or platform code, the content of the logs, or the monitoring metrics. Although our approach is non-intrusive, it depends on information from operation's logs and monitoring metrics. We assume that having higher-quality logs and metrics can improve the quality of anomaly detection. It can also help to improve the resiliency built into operations. For instance, we observed several cases where the rolling upgrade process attempted to terminate instances that had already gone out of service. This finding gave us an understanding about two limitations of this operation: (i) the operation did not check the status of the instance before attempting to de-register and terminate the instance; and (ii) the unavailability of the instance was not logged. These insights can be used to improve the rolling upgrade operation and its logging.

Another insight we gained was related to the process of collecting monitoring data. Collecting monitoring data for resources on a large scale and for 24 hours a day can be a costly process, and so it is important to collect monitoring data efficiently. Our case study and analysis showed that integration of context of the operation's behaviour

from logs with resource metrics can reveal which metrics we need for anomaly detection. To this end, we anticipate that adopting our approach to correlate an operation's behaviour derived from logs with monitoring metrics can also help to improve management of DevOps operations in the following ways:

- to understand the limitations of log content, and thus to improve the quality of the logging where needed;
- to have statistical information about the importance of metrics, and if the given frequency of monitoring data is sufficient; and
- to derive new requirements for improving operation processes.

7. RELATED WORK

There are two main areas of work related to our approach: error detection and 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. [32] 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 [15, 32]. Adopting any of these techniques comes with limitations. For example, rule-based techniques require large knowledge bases that are difficult to maintain, whereas model-based techniques require a detailed understanding of the system. Statistical techniques have been widely used for anomaly detection in system monitoring [15, 31, 33]. Most of the existing work in this domain focuses on changes in non-contextual data points (CPU utilization, memory usage etc.) and raise an alarm when there are breaches of thresholds. Our research is not focused on normal operation of a system, where such thresholds have wide applicability. Instead, we are specifically interested in anomaly detection during running DevOps operations. For anomaly detection during such sporadic operations, state based metrics like VM start and VM termination are most suitable. Existing literature is very rich in anomaly detection with data points; however, there have been few studies on using context (like log events in our work) and behavioral information for anomaly detection [15].

Most past approaches that use contextual information are appropriate for offline assessment, rather than for online assessment. Other approaches are mostly intrusive, that is, they require changes to be applied to the system. POD-Monitor [14] 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. POD-Monitor considers the operational context on the level of whole operation processes – e.g., rolling upgrade is running – and focuses on anomaly detection on resources, whereas we conduct anomaly detection at the fine-grained level of individual steps of operations. Another approach that addresses the above limitations, in part of one of the authors' previous work, is 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, 34]. This technique addresses the problem of online validation of operations to some degree. However, the approach has two limitations, which we discuss below: it requires manual assertion specification, and it relies on logs as the primary source of information.

The first limitation is related to manual assertion specification. Assertions in POD-Diagnosis check if the actual state of a system corresponds to the expected state of a system. In 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 time-consuming and thus, with fast evolving changes of modern applications, might not be practical. Second, manually specified assertion might not correctly express the exact timing and effects of a logged event, resulting in an imperfect specification and thus lowered precision in the assertion specifications. Third, manual assertion specification relies on the expertise of the administrator or developer writing it. If that developer is not the involved in developing the underlying tool, the expertise about the exact function of that tool is typical limited, 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=2,000 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.

The second limitation is related to the dependability on logs as the main source of information for operation monitoring. First, logs are often low-level, noisy, and with inconsistencies in style [35]. For instance, in [35] authors report the difficulties of failure detection due to logs having a lack of relevant information, and [36] highlights that over 60 percent of failures in their experiments of fault injection were not reported in the logs. Many of the current practices of generating logs focus on developer needs

during development time, rather than considering administrative needs in production settings [9]. Second, logs are voluminous, and it is usually difficult to derive which log line, or which set of log lines, is actually responsible for an action in changing the 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 due to frequent changes in large-scale applications, in which hundreds of shared resources are involved and resources are exposed to changes from multiple concurrent operations. Thus, it is not trivial to isolate the execution of one such operation from the other running operations. These limitations exacerbate the difficulty of error detection for cloud operations, and relying on log content limits the generalisability to tools with high-quality log output. 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. To tackle these limitations, this study attempted to leverage cloud metric data, in addition to information extracted from logs, to cross-validate the execution of cloud DevOps operations.

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 or troubleshoot unknown issues. Monitoring data can be collected through automatic calls of APIs or even streamed to outside services in near realtime. 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 [37]. One of the highlighted issues in this domain is the lack of cross-layer monitoring [37]. 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 types of monitoring information, which can span multiple layers.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have addressed the problem of monitoring cloud application operations through log and metric analysis. Our contribution includes a novel approach that assists in finding the subset with the most relevant monitoring metrics. It further includes employing those metrics for the reliable assurance of the correct execution of sporadic cloud operations which are common practice in DevOps, particularly our use case of staged upgrading of clusters of virtual machines (VMs). Core to this approach is a domain-agnostic regression-based correlation analysis technique that correlates operations' event logs and resource metrics. Based on this correlation, we can identify which monitoring metrics are significantly affected by an operation's activities and how. We illustrated that the selected target monitoring metrics, along with the derived regression model, can be used as the basis for generating runtime assertions which are suitable for the detection of anomalies in running operations. Further, we showed a method to distinguish alarms generated as the result of direct effects of an error from the ripple effects of errors. 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, respectively.

We demonstrated the applicability of our proposed approach in a comprehensive case study of rolling upgrade operation with different configuration here, however, we aim to conduct further experiments to assess the generalisability of approach for different operational environments. Moreover, we would like 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 an error 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.

References

[1] S. Elliot, DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified, Tech. rep., International Data Corporation (IDC) (Dec. 2014).
URL http://devops.com/blogs/real-cost-downtime/

[2] D. Cappuccio, Ensure cost balances out with risk in high-availability data centers, Tech. rep., Gartner (Jul. 2013).
URL http://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime

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

[4] Veeam, Veeam data centre availability report, Tech. rep., Veeam Software (Dec. 2014).
URL http://go.veeam.com/2014-availability-report.html

[5] Ponemon, Breaking down the cost implications of a data center outage, Tech. rep., Ponemon Institute (Dec. 2013).
URL http://www.emersonnetworkpower.com/en-US/Solutions/infographics/Pages/Cost_Implications_of_Outages.aspx

[6] Infonetics, The cost of server, application, and network downtime, Tech. rep., Infonetics Research (Jan. 2015).
URL http://www.infonetics.com/pr/2014/Cost-Server-Application-Network-Downtime-Survey-Highlights.asp

[7] O. Crameri, N. Knezevic, D. Kostic, R. Bianchini, W. Zwaenepoel, Staged deployment in mirage, an integrated software upgrade testing and distribution system, in: Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07), Stevenson, Washington, USA, 2007, pp. 221–236.

[8] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, A. D. Satria, What bugs live in the cloud? A study of 3000+ issues in cloud systems, in: Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 2014, pp. 7:1–7:14.

[9] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. Jain, M. Stumm, Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems, in: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI, Broomfield, CO, USA, 2014, pp. 249–265.

[10] X. Xu, L. Zhu, I. Weber, L. Bass, D. Sun, POD-Diagnosis: Error diagnosis of sporadic operations on cloud applications, in: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '14), Atlanta, GA, USA, 2014, pp. 252–263.

[11] R. J. Colville, G. Spafford, Configuration management for virtual and cloud infrastructures (May 2013).
URL http://goo.gl/P5edKP

[12] T. H. Nguyen, J. Luo, H. W. Njogu, An efficient approach to reduce alerts generated by multiple IDS products, International Journal of Network Management 24 (3) (2014) 153–180.

[13] A. Sadighian, J. M. Fernandez, A. Lemay, S. T. Zargar, ONTIDS: A highly flexible context-aware and ontology-based alert correlation framework, in: Proceedings of the 6th International Symposium on Foundations and Practice of Security, FPS, La Rochelle, France, 2013, pp. 161–177.

[14] X. Xu, L. Zhu, M. Fu, D. Sun, A. B. Tran, P. Rimba, S. Dwarakanathan, L. Bass, Crying Wolf and Meaning It: Reducing false alarms in monitoring of sporadic operations through POD-Monitor, in: Proceedings of the 1st IEEE/ACM International Workshop on Complex Faults and Failures in Large Software Systems, Florence, Italy, 2015, pp. 69–75.

[15] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv. 41 (3) (2009) 15:1–15:54.

[16] M. Farshchi, J.-G. Schneider, I. Weber, J. Grundy, Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis, in: Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering (ISSRE '15), Gaithersburg, Maryland, 2015, pp. 24–34.

[17] X. Xu, L. Zhu, D. Sun, A. B. Tran, I. Weber, L. Bass, Error diagnosis of cloud application operation using bayesian networks and online optimisation, in: Proceedings of the 11th European Dependable Computing Conference, EDOC, Paris, France, 2015, pp. 37–48.

[18] T. A. Limoncelli, S. R. Chalup, C. J. Hogan, The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems, Vol. 2, Pearson Education, 2014.

[19] J. Atwood, Working with the Chaos Monkey (Apr. 2011).
URL http://blog.codinghorror.com/working-with-the-chaos-monkey/

[20] Netflix, Chaos Monkey (Jan. 2015).
URL http://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

[21] D. G. Feitelson, E. Frachtenberg, K. L. Beck, Development and deployment at Facebook, IEEE Internet Computing 17 (4) (2013) 8–17.

[22] J. Miranda, How Etsy deploys more than 50 times a day (Mar. 2014).
URL http://www.infoq.com/news/2014/03/etsy-deploy-50-times-a-day

[23] T. Dumitras, P. Narasimhan, Why do upgrades fail and what can we do about it?, in: Middleware 2009, ACM/IFIP/USENIX, 10th International Middleware Conference, Urbana, IL, USA, November 30 - December 4, 2009. Proceedings, 2009, pp. 349–372.

[24] A. Wolski, K. Laiho, Rolling upgrades for continuous services, in: Service Availability, First International Service Availability Symposium, ISAS 2004, Munich, Germany, May 13-14, 2004, Revised Selected Papers, 2004, pp. 175–189.

[25] E. T. Roush, Cluster rolling upgrade using multiple version support, in: 2001 IEEE International Conference on Cluster Computing (CLUSTER 2001), 8-11 October 2001, Newport Beach, CA, USA, 2001, p. 63.

[26] I. Weber, C. Li, L. Bass, X. Xu, L. Zhu, Discovering and Visualizing Operations Processes with POD-Discovery and POD-Viz, in: Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '15), Rio de Janeiro, Brazil, 2015, pp. 537–544.

[27] R. J. Hyndman, G. Athanasopoulos, Forecasting: Principles and Practice, OTexts, 2014.

[28] J. W. Osborne, Prediction in multiple regression, Practical Assessment, Research and Evaluation (PARE) 7 (2) (2000) 1–9.

[29] D. C. Montgomery, E. A. Peck, G. G. Vining, Introduction to Linear Regression Analysis, 5th Edition, Wiley, 2012.

[30] L. Bass, I. Weber, L. Zhu, DevOps - A software architect's perspective, The SEI series in software engineering, Addison-Wesley, 2015.

[31] A. Patcha, J. Park, An overview of anomaly detection techniques: Existing solutions and latest technological trends, Computer Networks 51 (12) (2007) 3448–3470.

[32] S. Kavulya, K. R. Joshi, F. D. Giandomenico, P. Narasimhan, Failure diagnosis of complex systems, in: K. Wolter, A. Avritzer, M. Vieira, A. P. A. van Moorsel (Eds.), Resilience Assessment and Evaluation of Computing Systems, Springer, 2012, pp. 239–261.

[33] C. Wang, K. Viswanathan, C. Lakshminarayan, V. Talwar, W. Satterfield, K. Schwan, Statistical techniques for online anomaly detection in data centers, in: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, IM, Dublin, Ireland, 2011, pp. 385–392.

[34] X. Xu, I. Weber, L. Bass, L. Zhu, H. Wada, F. Teng, Detecting cloud provisioning errors using an annotated process model, in: Proceedings of the 8th Workshop on Middleware for Next Generation Internet Computing, MW4NextGen, Beijing, China, 2013, pp. 5:1–5:6.

[35] A. J. Oliner, A. Ganapathi, W. Xu, Advances and Challenges in Log Analysis, Communications of the ACM 55 (2) (2012) 55–61.

[36] M. Cinque, D. Cotroneo, R. Natella, A. Pecchia, Assessing and improving the effectiveness of logs for the analysis of software faults, in: Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010, Chicago, IL, USA, June 28 - July 1 2010, 2010, pp. 457–466.

[37] G. Aceto, A. Botta, W. de Donato, A. Pescapè, Cloud monitoring: A survey, Computer Networks 57 (9) (2013) 2093–2115.