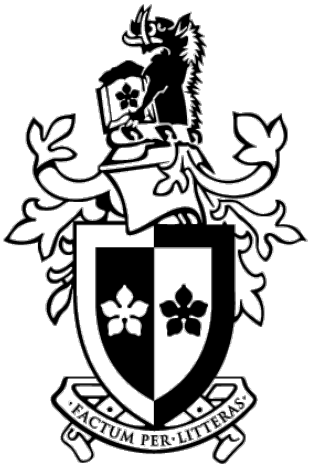
Anomaly Detection Using Logs and Metrics Analysis for System Application Operations

Mostafa Farshchi

Submitted in fulfilment of the requirements of the degree of **Doctor of Philosophy**



Faculty of Science, Engineering and Technology

Swinburne University of Technology

Melbourne, Australia

February, 2018

Abstract

The dependability of systems and software has been a topic of research for at least the last five decades. Despite the usage of the modern technologies and the advances in system monitoring and anomaly detection techniques, failure rates are still high. Cloud computing systems are a prime example of the usage of such technologies that provide facilities 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-related failures have been reported as one of the leading causes of system-wide outages.

This specifically applies to Development and Operations (DevOps) processes, such as backup, redeployment, upgrade, customised scaling, and migration – which are executed at much higher frequencies now than a decade ago. These frequent changes in environments with virtualisation technologies, that are inherently complex due to the flexibility provided and a large number of resources involved, make monitoring and anomaly detection of applications and resources a challenging task.

One of the key challenges in current anomaly detection techniques is to perform anomaly detection with regards to the type of activities or the context that a system is exposed to. Another challenge with anomaly detection in modern environments such as cloud computing is the large scale of resources involved which can lead to a large volume of monitoring data. Also, there is a challenge related to having combined cross-level monitoring from various resources such as logs and resource metrics. Therefore, this thesis focuses on addressing the above challenges with a focus on system and application operation health and performance monitoring.

In this thesis, we propose a novel approach to employ both application operation logs and resource metrics data for near real-time anomaly detection. We propose a set of methods to map the contextual log events to observed data from resource metrics. Also, this thesis presents an effective solution to identify the metrics that are most relevant to operational behaviour and shows how the metric selection leads to better anomaly detection. In addition, we propose an approach to derive a correlation model between logs and metrics and leverage the model to derive assertion specifications. With these assertions, we do not only detect anomalies at run-time, we also know which types of log events as well as which resources are involved in an anomalous incident.

The anomaly detection approach proposed in this thesis is a non-intrusive, unsupervised, context-based anomaly detection technique. We have conducted a set of experiments with different configurations for two industry-grade case studies. In this evaluation, our approach was effective in detecting anomalies caused by faults with high accuracy.

Acknowledgements

No words can express my gratitude to my PhD supervisors, Jean-Guy Schneider, Ingo Weber, and John Grundy. Without their continuous guidance, critical feedback, and encouragement, this thesis would not exist. Jean-Guy, your seasoned supervision, helpful advice, and insightful feedback has been invaluable for me. I am very thankful that you have always been willing to take extra time out of your busy schedule when an important issue came up. I learned a lot from your guidance, especially your structural and methodological analysis, and attention to detail. Ingo, I am very grateful for all your guidance and support, especially in that you have been very meticulous about my writing and providing constructive feedback. Also, your regular attendance at our weekly meeting through video conferencing and other times through Skype or phone calls gave me the feeling that I could reach you at any time. John, I am very glad and thankful for, firstly, suggesting that I apply for a PhD at Swinburne and introducing me to Jean-Guy, and secondly for providing your advice and feedback throughout my PhD journey.

I would also like to to thank my friends, who have made this difficult journey easier for me. I thank Amir Kiyaei for all his empathy and encouragement, which has been valuable for me. Also, I like to thank my friends Amin Rigi and Sahar Sohrabi, who have helped me tremendously in this journey through their affection, feedback, and company, from our English writing practice, to our research discussions and relaxing tea breaks.

I thank the Swinburne University of Technology and Data61, CSIRO, for providing funding and facilities to undertake this research. In particular, I would like to thank my colleagues at the Process Oriented Dependability Group in Data61, who gave me the chance to make use of their consultation, and to

access their tools and data. Special thanks go to An Binh Tran, Sherry Xu, Liming Zhu, and Len Bass. Also, I would like to express my appreciation to Raffaele Della Corte, Marcello Cinque, and Antonio Pecchia from Università degli Studi di Napoli Federico II, who provided access to the data of one of the case studies of this thesis. I also like to thank Audrey van Ryn for helping with proofreading and the thesis examiners for providing their valuable comments and feedback.

Last but certainly not least, I am deeply grateful to my family for all their support, affection, and love. I am especially indebted to my mother, who had to bear a difficult time of sickness while I was far away from her. I express my deepest love and regard for my mother.

Declaration

This is to certify that this thesis contains no material which has been accepted for the award of any other degree or diploma and that to the best of my knowledge this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Mostafa Farshchi

February, 2018

List of Publications

1. Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, John Grundy: Metric Selection and Anomaly Detection for Cloud Operations using Log and Metric Correlation Analysis, *Journal of Systems and Software*. Mar 2017.

2. Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, John Grundy: Experience report: Anomaly detection of Cloud Application Operations Using Log and Cloud Metric Correlation Analysis, *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Washington, D.C., USA, Nov 2015, pp. 24-34. **(Best Paper Award)**

3. Ingo Weber, Mostafa Farshchi, Jan Mendling and Jean-Guy Schneider: Mining Processes with Multi-Instantiation, *ACM/SIGAPP Symposium on Applied Computing (ACM SAC)*, Salamanca, Spain, April 2015.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The failure of systems and software has been a topic of research for a very long time [40, 48, 68]. Despite many advances in this area failure rates are still considerable [24]. 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 the International Data Corporation (IDC) [35] 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 [22], Avaya [7], Veeam [123], and Ponemon [106]. These surveys estimate the cost of application downtime to be 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/or network downtime, respectively [61]. Such significant losses (both in monetary and non-monetary terms such as losing customers trust) 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 [30, 31, 51, 143], and an empirical study conducted by Yuan et al. [143] reports that *operational activities* such as backup, update and upgrade, and migration 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 and large-scale applications, especially in the

situations where system virtualization is used such as in cloud environments. Virtual machines facilitate the hosting of software application services in an isolated way that simulates a physical computer system. In such environment, there is a significant range of dynamism and flexibility provided. A user can increase or decrease the dedicated resources to a VM or change the number of available VM instances that are in service, where resource dedication can be automatically configured and adjusted by observation of workload or detection of anomalies in a system [15]. These environments are inherently complex due to the flexibility provided and a large number of resources involved. Detecting errors in these types of environments has always been challenging and has attracted a number of studies [15, 50, 67, 68, 78]. Modern applications are subject to regular changes from sporadic operations, for instance, operations such as on-demand scaling, upgrade, migration, and/or reconfiguration [140] cause frequent changes to the systems and applications.

Given these complexities, it is not surprising that detecting and diagnosing operational-related failures have been reported as one of the main challenges in system failures and outages [30, 140]. However, operation and configuration activities have not received the much-deserved attention until the recent emergence of the software DEVelopment and information technology OperationS (DevOps) movement.

DevOps is a concept that emphasizes on collaboration and communication between software developers and system administrators. Bass et al. defines DevOps as "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality" [12]. In another definition, Hüttermann defines DevOps as "practices that streamline the software delivery process, emphasizing the learning by streaming feedback from production to development and improving the cycle time" [57].

As the result of DevOps, unlike the past that major changes to the complex system were infrequent and often done during scheduled downtime, in modern complex systems, various sporadic operations are conducted with a high frequency [110]. These operations involved in *continous deployment* practices

take place from few times a month to over 1,000 deployments into production each day [110].

Sporadic operations are usually implemented by a set of separate tools and are subject to interference from simultaneous operations dealing with the same resources. For instance, an application service composed of multiple Virtual Machine (VM) instances may go through a routine backup operation from a commercially developed tool while the same system may also be exposed to an either manual or automatic scaling up or scaling down operation. Such interferences of operations on same resources make the identification of abnormalities of system resources a highly challenging task. Executing an operation like an 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.

To this end, this thesis has the aim to find mechanisms to improve dependability [1] assurance of systems exposed to sporadic operations, especially systems with virtualization technologies such as in cloud environments. In this direction, the research problem is presented in the next section. Then, the objective of this research and research questions are presented. Next, an overview of the thesis structure is given. Finally, research contributions are listed, followed by the list of related publications where the contributions of this research have in part been published.

1.1 Research Problem

One way to improve a system's dependability is to leverage a set of tools and techniques to monitor running operations and to assess their impact on a system in real-time and detect anomalous states. However, many challenges are associated with monitoring of application operations that makes the task of dependability assurance of application operations a very challenging task.

One challenge is that system operators have to deal with tracking multiple monitoring metrics and may receive too much monitoring information, includ-

[1] This thesis mainly addresses the reliability and availability aspects of system dependability according to the definition given in [8]. We will provide definitions and background in more details in Chapter Two.

ing many false warnings and alerts. Ongoing system monitoring of systems with multiple resource nodes can lead to an overwhelming volume of monitoring data [50]. This distracts system operators from becoming aware of critical abnormal situations [94, 109]. This problem has caused operators to disable monitoring when sporadic operations are running, so as to avoid too many false alerts [50, 138].

The next limitation is related to the type of anomaly detection. Most of the existing approaches to systems monitoring and anomaly detection solely focused on *point-based* monitoring: they observe the state of hardware and software metrics, such as CPU utilisation, network traffic, etc. without monitoring the *contextual behaviour* of a system or inspecting the impact of activities of application operation on systems resource utilization. In the past, as the sporadic operations were not as frequent as today, this did not lead to many anomalous states, but with the frequent operations like continuous deployment in modern complex application systems, this can lead to many anomalous situations and, thus, generation of too many false positive alarms.

Another challenge is related to log monitoring. When it comes to monitoring with consideration of system behaviour, an application operation's log is the primary source of information for monitoring the *system's behaviour* [28, 96]. Logs provide valuable information about running operations and they have been employed for system failure detection and diagnosis in the past [95], yet they are not fully reliable as there are various limitations in monitoring by system logs [23]. Logs are usually low-level, noisy, and lack the information about changes to the states of resources. This limits the usefulness of logs as the main source of information for system behaviour monitoring [130, 140]. Many studies have shown that some of the failures and abnormalities that happen in a system either are not detected and reported by log events, or they are detected in various degrees, depending on the quality of the logging mechanism of the system under test [28, 29].

Moreover, many of the current techniques of anomaly detection in a domain of system health and performance monitoring are supervised methods, which require *labelled data instances* for both normal and anomalous items [60].

Methods working with the labelled data have an underlying assumption that there is a priori knowledge for each data instance of being normal or anomalous data. However, obtaining data with anomalous data labels is often very challenging and when anomalous data instances are available, their frequency is rare [23]. Therefore, one important challenge is to employ unsupervised anomaly detection techniques that do not require labelled data.

Last but not least, major monitoring and anomaly detection approaches merely rely on one source of information for their monitoring purposes, while none of the monitoring data from metrics or logs alone can cover both aspects of the behavior and the status of the system. The need of cross-level monitoring has been highlighted in the literature [2], yet how to integrate various sources of monitoring information is not well addressed. Therefore, one important challenge is to combine and map different sources of monitoring data for better detection of errors and abnormalities in a system, respectively.

Above-mentioned issues and challenges make dependability assurance of system and application operation health and performance monitoring a very challenging task. Yet, these issues have not been well addressed, especially in regards with employing unsupervised context-based methods. Therefore, in this thesis, we aim to address these matters with a focus on application operations and system health and performance monitoring.

1.2 Objective and Research Questions

The aim of this thesis is as follows:

To design, implement and evaluate mechanisms to improve dependability assurance of system application operations through monitoring of system resources and activities, as well as anomaly detection techniques.

Given the problems highlighted in the previous section and to fulfil the above objective, the main research questions that this thesis aims to answer are:

- **RQ1:** As event logs and resource metrics are two separate sources of

monitoring information with different structures, how can we combine the information from these two sources and derive a statistically meaningful relationship between them?

- **RQ2:** What is an appropriate mechanism to distinguish insensitive monitoring metrics from the ones that are sensitive to the activities of system application operations and how do we best leverage this to identify the most suitable monitoring metrics for anomaly detection?

- **RQ3:** How can we build a statistically supported model based on log and resource metrics data to derive assertions that enable high-accuracy, unsupervised, contextual anomaly detection?

1.3 Research Contributions

This thesis makes original contributions to the knowledge in the areas of system dependability by focusing on application operations and system health and performance monitoring. In this direction, this thesis aims to develop a novel approach to address the difficulties related to system monitoring and anomaly detection that are exposed to sporadic operation through learning from the correlation between changes on the status of resources and activities of application operations.

The main overall contribution of this thesis is twofold: first is a metric selection mechanism based on the relationship between system resources and activity logs; second is an unsupervised contextual-based anomaly detection approach by combining both sources of monitoring information that show the behaviour of the systems through activities reported in event logs and status of resources in metrics.

Therefore, this thesis aims to contribute to system dependability by proposing solutions that take the dynamics of system behaviour into account along with states of resources for system monitoring and anomaly detection. The first step to fulfil this aim is to find a way to map and combine the two sources

of information of logs and metrics. However, the data collected from contextual logs are essentially different from the metrics collected from the status of resources. Logs are textual information while metrics are often a set of numerical values. How to map and combine these two sources of information was one of the key challenges of this thesis.

This thesis proposes a novel approach to successfully employ both types of information for real-time anomaly detection of a system and also to use the proposed approach for verifying the correct execution of the steps of an operation. In fact, the proposed method is capable of detecting at what step of a process an anomaly occurred. Such outcome can be employed for dependability assurance of critical operations like back-up, upgrade, and migration processes.

To this end, the proposed approach improves the dependability assurance of systems through the following key contributions:

1.3.1 Logs-Metrics Mapping

The data collected from contextual logs often have a different format from the numerical metrics gathered from observing the status of resources. To map and combine these two sources of information we proposed the following methods:

- **Representing logs as quantitative metrics:** operational logs are often fine-granular and voluminous - the number of logs in a typical application operation may include thousands log lines [95, 119]. Logs are being broadly adopted to record runtime behaviour of software systems [54], so to know how the behaviour of a software system changes the states of resources it is necessary to find a way to track log events. Therefore, we first adopt an approach that uses pattern matching using *regular expressions* to identify all log events that have similar patterns and label similar log events as a unique log event type. Then we propose a mechanism to represent each log event based on the interpolated occurrence strength of the log event type that occurs within a specific time-window (e.g. one-minute time-window). The interpolated weight-timing approach proposed generates a metric that is indicative of what

type of log event occurred, how many times it occurred in a time-window, and in what relative interval that log appeared. The proposed extracted interpolated occurrence strength allows us to present logs in a quantitative form which can be used for statistical analysis and it enables us to map the event log into the same time-window interval of available monitoring data of resource metrics.

- **Clustering low-granular logs using weight timing and correlation coefficients:** Most often system behaviour is not characterised by a single log event; it is often a set of log events indicate changes in the state of a resource. Therefore, it is important to identify which set of log events are responsible for certain changes in a system, which in this thesis it is referred to as *log abstraction*: the process of clustering individual correlated log events to log activities.

  We propose a log abstraction technique to cluster low-granular relevant log events into a set of meaningful activities. The proposed approach in this thesis uses the combination of weight timing within a time-window and Pearson correlation coefficient analysis to address this issue. We demonstrate how this approach is effective on utilising the output of metric extraction from the previous step along with Pearson coefficient correlation analysis to produce a set of meaningful log activities. The results of the produced log clusters are compared with a previous related study that used experts judgement to cluster log event - our study showed our automatic approach produced comparable results to the manual outcomes from experts reviews.

- **Mapping logs to metrics and identifying influential log activities:** A statistically based solution with employing regression analysis is presented to derive a model to explain the relationship between log messages and metrics values. There is a large number of studies in anomaly detection using statistical techniques and few of them have used regression analysis [23, 60]. However, most of these studies are limited to *point-based* methods rather than context-based methods. With

context-based methods, concurrent utilisation of logs and metrics has not been well studied in an integrated form for fault detection and diagnosis. To the best of our knowledge, this thesis is the first work that proposes a systematic approach that attempts to perform a statistically supported exploratory analysis between log events and resource metrics using regression analysis and identifies the individual log activities that are responsible for changes on system resources utilisation.

1.3.2 Metric Selection

This thesis proposes a systematic approach to find the resource metrics that are most sensitive to operations' behaviour. The proposed approach addresses the current challenge of dealing with too many monitoring metrics to identify the most suitable metric candidates from employing regression analysis and then ranking the metrics based on their lowest to highest sensitivity to the behaviour of the operation.

In this thesis, we present a method that is effective in finding the metrics that are most relevant to operational behaviour and show how the statistical-based selection of metrics lead to better anomaly detection of a system application operation. In particular, our contribution to reducing monitoring metrics dimension bring the following benefits:

- By significantly reducing the metric dimensions to the essential ones, it helps the operators to focus on a limited number of metrics rather than a long list of metrics.

- Large volume of monitoring data, too many false positive alarms, and several alarms from the same event have been reported to overwhelm system operators cause alarm fatigue. Reducing metrics dimension can be helpful in this matter [23, 50, 138].

- System monitoring and anomaly detection are often computationally intensive [20], and many approaches have been suggested to reduce this cost, especially for large-scale infrastructure by reducing precision [62], or employing adaptive monitoring [67, 91] and decentralized architectures

[120] or other methods [2, 38]. Our approach, in particular, contributes to this domain through reducing the cost of system monitoring by narrowing down the focus on the most relevant metrics.

1.3.3 Assertion Specification Derivation and Anomaly Detection

Another important contribution of this thesis is related to defining assertion specifications from the regression analysis. We propose a method which uses the outcome of identified monitoring metrics to derive a statistically justified mathematical model that can be used for run-time assertion checking. *Assertions* are used to check if the actual state of the system corresponds to the expected state of the system; otherwise, the data instance is detected as an anomaly. The expected state of the system is derived from log analysis using assertion specification derivation.

In this process, the selected metrics are employed at run-time 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 outcome of assertion prediction at run-time. This assertion checking is a crucial part of error detection in monitoring the execution of operation steps, especially for critically sensitive operations like migration, upgrade, security checking etc.

With the help of assertion specification, we detect anomalies at run-time, we also know which types of log events as well as which resource are involved in an anomalous incident. Also, our approach is a non-intrusive solution, meaning that it does not require changes to the source code or the logs. Although the regression analysis has been employed for anomaly detection, to the best of our knowledge, this is the first work in this domain that derives assertion specification from regression analysis with considering the context of logs and metrics at the same time for a non-intrusive and online anomaly detection method.

1.3.4 Ripple Effect Detection

In this thesis, a method is proposed to identify anomalies resulting from ripple effects of errors and distinguishing them from direct effects of errors. When a failure occurs in a system, it often propagates multiple anomalous symptoms that are all caused by one error but they may be observed at different points in time, which can trigger too many false alarms [138]. Also, with the plethora of monitoring and security tools that have overwhelmed the system administrators with an exponentially growing number of information, excessive alarms may make operators to miss the crucial vulnerabilities [46].

In this direction, the proposed method contributes to this domain by distinguishing the alarms that are identified as the result of direct anomalies from the ones that are the ripple effects of the already reported anomalies. By differentiating these two, we can reduce the redundant false alarms and as the result to increase the precision of anomaly detection. In the algorithm proposed, the ripple effects are reported as warnings rather than actual errors.

1.3.5 Anomaly Detection Evaluation

To conduct experiments and facilitate the anomaly detection process, a service-oriented tool is developed to check system anomaly in real-time. This tool gets the input of the regression analysis from configuration settings and checks automatically if an anomaly occurs at run-time.

On the basis of this tool, the ideas proposed in this thesis are evaluated with two separate industry-grade case studies. We have conducted experiments to evaluate the proposed approach in one case study of using Rolling Upgrade operation with different configuration settings and upgrade two, eight, and forty virtual machines on the Amazon AWS public cloud service. In addition, we explored and investigated the applicability of our approach with data from an air traffic middle-ware system.

These two case studies have significant differences from each other including the environment, the scale of logs and metrics, the forms of metrics available, and the type of faults injected. In these case studies, we used our approach to abstract fine-granular input logs to a set of coarser-granular activities, map

textual logs to numerical metrics, select most relevant target metrics, learn from error-free traces, specify assertions, and evaluate if the assertions can detect injected faults.

1.4 Organisation of the Thesis

The organisation of the thesis is outlined below:

Chapter 2 provides the necessary background of the topic of interest of this thesis and overviews the related work that addresses problems similar to the one considered in this thesis. In particular, this chapter provides a background and literature survey on system health and performance monitoring, metric selection, and anomaly detection.

Chapter 3 summarises and formalises the problem of the work and presents the proposed conceptual framework and methodology to address the research problem of the thesis. In this chapter, the steps of the proposed approach including the steps of obtaining data, performing log analysis, finding relevant metrics from all available metrics, and the process of assertion derivation for anomaly detection are explained and elaborated.

Chapter 4 gives an overview of the tools used in this research and illustrates the prototypical implementation of the tool to conduct and evaluate our proposed online anomaly detection approach.

Chapter 5 investigates the applicability of the proposed solution, as implemented in the tool, with a comprehensive case study of Rolling Upgrade operations. The steps described in our methodology from chapter 3, are further investigated, and evaluated with this case study. The conducted experiments provide a practical example of challenges faced in the process of upgrading systems in a real cloud environment. Further, the effectiveness of the approach is evaluated by injection of random faults and assessing the success rate of accurate anomaly detection.

Chapter 6 assesses the applicability of the proposed approach in a second case study. Similar to the Chapter 5, this chapter also elaborates the steps taken to evaluate the applicability of the proposed approach for the data

obtained from a case study with an air-traffic middle-ware system.

Chapter 7 concludes the thesis by summarising the presented results and their contributions to the knowledge, answering the research questions, and highlighting possible extensions and future work.

1.5 Related Publications

The content, material and results presented throughout this thesis have been partially published and presented in the following papers:

- Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, John Grundy: Metric Selection and Anomaly Detection for Cloud Operations using Log and Metric Correlation Analysis, *Journal of Systems and Software*. Accepted Mar 2017.

- Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, John Grundy: Experience report: Anomaly detection of Cloud Application Operations Using Log and Cloud Metric Correlation Analysis, *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Washington, D.C., USA, Nov 2015, pp. 24-34. **(Best Paper Award)**.

- Ingo Weber, Mostafa Farshchi, Jan Mendling and Jean-Guy Schneider: Mining Processes with Multi-Instantiation, *ACM/SIGAPP Symposium on Applied Computing (ACM SAC)*, Salamanca, Spain, April 2015.

Chapter 2

Background and Related Work

In this chapter, we provide background information for the research problem addressed in this thesis. First, we provide definitions for the key terms that are frequently used throughout the thesis. Then, we present the background of the concept of sporadic operations and explain some of the current challenges of systems operations monitoring, in particular, for the domain of monitoring systems with virtualization technologies.

In addition, we conduct an analysis of contemporary metric selection and anomaly detection techniques, and highlight the differences of our approach in comparison to the existing methods.

2.1 Definitions

This section provides clarification of some of the concepts that we will frequently refer to throughout this thesis.

The thesis aims to improve the dependability of system application operations through system logs and system resource monitoring. Many of the definitions used, including the definition of system dependability, are taken from the "Basic Concepts and Taxonomy of Dependable and Secure Computing" paper in [8]. According to this paper, dependability is an integrating concept that includes the following attributes:

- **availability**: "readiness for correct service"
- **reliability**: "continuity of correct service"

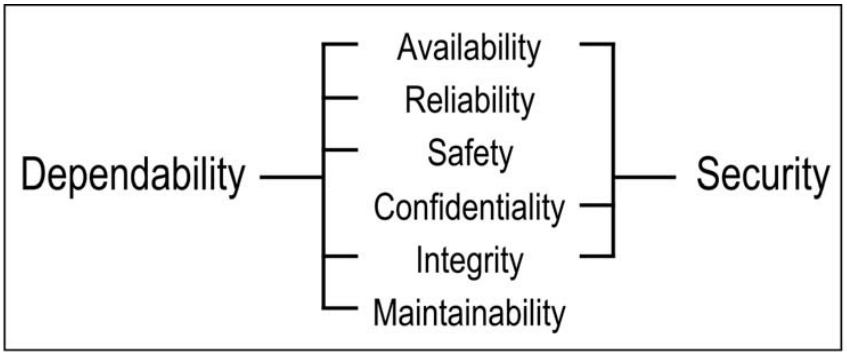


Figure 2.1: Dependability and security attribute [8].

- **safety**: "absence of catastrophic consequences on the user(s) and the environment"
- **integrity**: "absence of improper system alterations"
- **maintainability**: "ability to undergo modifications and repairs".

Among the above attributes, *security* requires the existence of **confidentiality**, integrity, and availability[8]. This relationship is shown in Fig. 2.1.

This thesis has a key focus on system health and performance monitoring, and anomaly detection. Thus, it mainly addresses the reliability and availability aspects of system dependability. Therefore, usage of the term "dependability" throughout the thesis refers to these two aspects of system dependability.

Also related to system dependability, we use fault, error and failure terms throughout the thesis. The relationship between fault, error, failure and the event log is depicted in Fig. 2.2. As the figure shows, an activation of a fault may lead to an error, and some of the errors may cause a system or a service failure, but not all failure and errors may be reported immediately in the monitoring mechanisms. For instance, a connection to a database may be lost and reported as an error in a log event, but a VM may go to a halt state and no errors are reported. More precise definitions of these terms are as follows:

- **failure:** "failure is an event that occurs when the delivered service deviates from correct service" [8].
- **error:** "an error is a state that may bring about a subsequent failure: a failure happens when an error reaches the service interface and alters the service" [8].

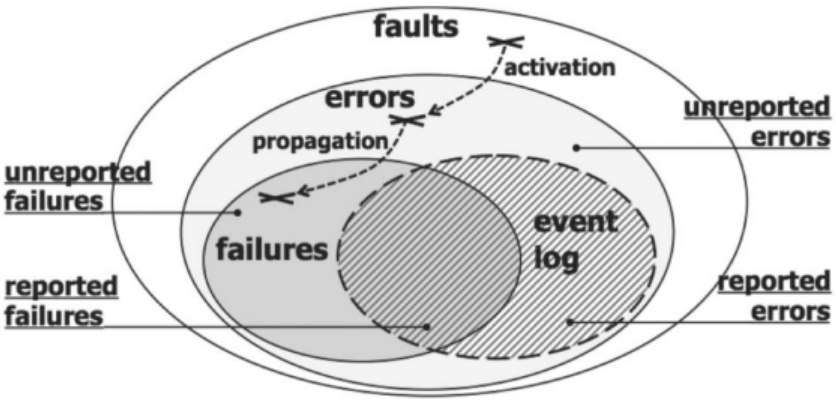


Figure 2.2: Relationship between faults, errors, failures and event log [27].

- **fault:** "The adjudged or hypothesised cause of an error is called a fault" [8]. A fault can be caused by a computation process, a specific sequence of inputs, or environmental conditions [104].

  event log: "A single line of text containing various fields (time-stamp, nodeID, protocol, application, error message) that reports a given activity of a system. Such an event is also often called a log message" [53]. It is worth to note that, a log event is the manifestation of an event, or a trace that an event has happened.

2.2 System Monitoring with Virtualisation Technologies

This thesis aims to improve system health and performance monitoring, with the focus on systems with virtualization technologies. These technologies have quickly gained popularity, and virtual resources have become broadly available. Virtual machines facilitate the modularity of application services and help to deliver services to a large number of service consumers by providing each with the illusion of a dedicated server. Customers[1] can rent VMs and automatically change their computational demands based on the pay-for-use concept in the cloud space. Virtualisation has become the mainstream practice for deployment of multiple application services in a multiple numbers of

[1] Organisations using virtualised resources to serve their end users

virtual machines running in one or a limited number of physical servers. In fact, virtualization has become the de facto standard of cloud-based services.

With the emergence of the virtualization environment, the dependability assurance of systems is not just limited to operating systems and physical resources; it also includes virtual machines, hypervisors and related technologies. Virtualization provides the flexibility of performing many operational tasks that have not been easily possible.

Examples of these tasks are cloning, scaling up or scaling down, changing the size of virtual machines, migrating of VMs, changing of inclusion and exclusion of various system components, changing execution environments, and frequent release and deployment [47, 64]. In this domain, system monitoring is not confined to few resources: system operators have to deal with a large number of metrics showing the status of resources, operations running, and applications of individual VMs.

In addition, multi-tenancy using virtualization technologies exacerbates the complexity and dynamism of this environment. This unprecedented dynamicity introduces new challenges in monitoring systems and operations, as the occurrence of a failure is common in such an environment.

The dependability assurance of VMs is of particular importance in cloud environments. 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 design them.

In contrast to service delivery in the cloud, where the status of a VM is important in an aggregated form, at the operational level (e.g., upgrade, deployment, or backup), each individual VM and its attached resources are important in their own right. From time to time a VM fails – for example, it 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 VM is running on, memory over-usage due to an increased system load, an application bug that stresses the VM, an operating system kernel bug, or random system termination for the assessing of a system's resiliency and recoverability in production. The occurrence of any of these failures during an upgrade operation can put the upgrade process on hold or even derail it.

Hence, we are interested in investigating the applicability of the approach of this thesis for anomaly detection for environments with virtualization architecture.

2.3 DevOps Operations

As highlighted above, environments with virtualization technologies are exposed to frequent changes by various operations, especially due to DevOps processes. DevOps is a concept that emphasizes collaboration and communication between software developers and system administrators, also defined as "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality" [12]. These types of system operations are also referred to as **sporadic operations** [79, 139, 140].

Sporadic operations refer to a subset of administrative operations that do not necessarily 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" [139]. Sporadic operations are often not running continuously or with predictable frequency, while often causing stress on resource consumption, thus many anomaly detection approaches mistake the effect of these operations on systems as anomalies [138].

Examples of sporadic operations are *Backup*, *(Rolling) Upgrade* (upgrading multiple virtual machines in an iterative way, such as updating 100 machines by updating 10 machines at a time), *Cloud migration* (migrating from one or a set of server types to another set of servers or also migrating to another cloud service provider), *Reconfiguration*, *On-demand scaling* (increasing or de-

creasing the number of machines in service in response to the computational demand, often done automatically with a set-up of predefined rules), *Rollback-/Undo* (changing the state of operational service to a healthy restore point in the case of detecting unexpected errors or failure, and *Deployment*.

Cloud sporadic operations can be 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 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 in Amazon AWS are: detaching or attaching an elastic block storage (EBL) disk volume from/to a VM; changes in conditions and configuration of an auto-scaling group (ASG), such as its size (horizontal scaling in/out); manual termination or reboot of a VM; VM 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 [47, 64]. They are another motivation for our work, demonstrating that the validation of sporadic operations has critical importance.

In the previous section, we discussed several technological challenges in using VM technologies. Sporadic operations have to deal with the above technologies, while these operations often have a system-wide impact (e.g. migration operation). Such complexities make the assurance of successful execution of this type of operation a challenging task. Therefore, within the scope of system health, performance monitoring and anomaly detection, we are interested in systems with virtualization technologies and DevOps operations.

2.4 Metric Selection

Continuous system health and performance monitoring of large-scale systems can lead to an overwhelming amount of data. The volume of system monitoring

data in a large-scale system can reach hundreds or even thousands of terabytes [97, 100]. Processing a large volume of data as the result of having many metrics can cause a significant increase in the cost of system monitoring [20, 67, 91].

In addition to the above, having a large number of metrics often increases the complexity of the process of anomaly detection [49]. Administrating tens or hundreds of machines while tracking changes on each metric is often impractical, and, in many cases, not immediately beneficial. In fact, system operators are often exposed to a plethora of monitoring information, and they receive too many monitoring warnings and alerts [94, 109, 138].

The current state of the practice of metric selection relies on system operators, with many alert configurations defined based on operators' domain knowledge, and sometimes alerts are set arbitrarily or based on ill-informed thresholds [90]. This exposure to a plethora of data makes system monitoring an overwhelming task for system operators.

To facilitate the above process, many software monitoring packages such as Nagios[2], Sensu[3], and Misto[4] provide customization, often allowing the choice of which metrics to show and how alarms should be generated. However, this customization has become more difficult with the ever-growing number of resource types in cloud environments [49].

In Chapter 1, Section 1.3.2, we highlighted the benefits of and reasons for metrics selection. In addition to the above, in anomaly detection for system health and performance monitoring, having the maximum number of metrics may not lead to better accuracy. On the contrary, it may degrade the accuracy of anomaly detection [49]. Most of the current anomaly detection solutions focus on anomaly detection techniques while paying little attention to the metric selection process.

Fu et al. [44] proposed an approach to quantify the relevance and redundancy of system performance metrics for cloud anomaly detection. The proposed approach first employs mutual information to measure the dependency

[2]https://www.nagios.org/
[3]https://sensuapp.org
[4]https://mist.io/

between two metrics, and then adopts principal component analysis (PCA) [1], as a feature extraction method, to extract a subset of metrics that best represent all metrics. In this approach, PCA-based metric extraction for cloud anomaly detection is based on the hypothesis that the "amplitude of performance metrics of anomalous system components increases as the severity of the fault/failure increases". Although this approach is shown to be effective for selecting metrics that are sensitive to failures, the metric selection extraction is bound to the availability of failure samples. In fact, this is a semi-supervised tree classifier approach for metric selection. This approach has the limitation of requiring labelled data for metric selection and anomaly detection.

In another study, Yang et al. [141] proposed a statistical method to diminish the volume of performance data. They used a Pearson product-moment correlation coefficient to obtain a quantitative estimation of the strength of correlation between performance metrics. Then, they adopted a stepwise regression-based method to identify the metrics that are relevant to application performance metrics. This approach relies on statistical techniques. However, this method does not take into account the selection of metrics with regard to the workload or activity logs.

Another example of the adaptation of statistical methods for identifying relevant metrics is presented in [65]. This paper models the quantitative relationship between the application performance and virtualized system metrics using regression analysis, and proposes a metric selection algorithm to rank the relevance of metrics from reconciled models related to application response time. This approach has the limitation of taking into account just the application response time and not considering the impact of operation activities.

Another line of work makes use of machine learning techniques. In this direction, Zhang and his colleagues [145] find that by using a tree-augmented naive Bayesian network (TAN) method, they can find which low-level system metrics are correlated to high-level metrics of service level objective (SLO) violations. Their approach is based on driving models from data using pattern recognition and probability modelling techniques. This approach has the advantage of being independent of domain knowledge; however, labelled data

(supervised learning) is needed to derive a metric attribution model. Also, the high computational complexity of TAN makes it unfit for online anomaly detection at large-scale. In this line, similar approaches have been employed for classification of a large number of performance metrics by creating "fingerprints" for observed performance anomalies via a selected subset of relevant metrics [17].

Although the selection of relevant metrics is vital and can directly impact the efficiency and accuracy of anomaly detection, and while feature selection techniques are being broadly used to reduce the number of features or variables in machine learning and statistics, very few studies have attempted to address the problem of metric selection in the domain of system health and performance monitoring [17, 44, 141, 145]. Therefore, in this study, we propose an approach for metric selection as part of our anomaly detection approach, and hence, in the current section, we reviewed the above studies that have explored this domain.

2.5 Anomaly Detection

In this section, we provide background on the concept of anomaly detection. Then, anomaly detection solutions will be discussed, based on the types of solution and the availability of data. Further, a classification of the most employed anomaly detection techniques in the domain of system health and performance monitoring are presented, along with their advantages and disadvantages.

We will review the current methods closely related to the proposed approach of this thesis after introducing our approach in the last section of Chapter 3. We will discuss the strengths and limitations of each method and compare the approach of this thesis with previous work.

2.5.1 What is Anomaly Detection?

The practice of detecting anomalies in a dataset is commonly referred to as the identification of items or events that do not match the expected patterns of other data in a dataset. "Anomalies are patterns in data that do not conform

to a well-defined notion of normal behaviour" [23]. Anomaly detection has application in a broad range of domains, such as in fraud detection, security breach detection, network intrusion detection, system monitoring and error detection, medical problems, or image and text processing. For example, in the medical domain, the anomalous data observation from an MRI image may show the presence of malignant tumours [113]. Anomalous activity in credit card transactions could be the indicator of credit card fraud [5]. Also, network traffic with an anomalous pattern may indicate an intrusion into the network [70].

Anomaly detection is different from noise detection, although they are often related. According to [23], "Noise can be defined as a phenomenon in data that is not needed to be analysed, but acts as a hindrance to data analysis". In other words, a type of irregularity in the data that is interesting to the analyst and has relevance to the nature of the data, which is often an indicator of an unexpected event or phenomenon, is referred to as an anomaly. In fact, noise removal is usually a pre-processing step of removing unnecessary items before data is used for the process of anomaly detection.

2.5.2 Anomaly Detection Characterisation: Data Labels

Anomaly detection processes can be classified from different perspectives, and a few past survey studies [3, 11, 23, 56, 85, 86, 102], have attempted to offer some classification in this domain. Availability of data labels is one of these classifications.

Data with labels refers to training data where the status of data records is known, whether these records have anomalous status or normal status [23]. Considering the availability of data labels, anomaly detection techniques are classified into three categories: supervised anomaly detection, unsupervised anomaly detection, and semi-supervised anomaly detection.

Supervised Anomaly Detection: Anomaly detection approaches under this category have the underlying assumption that training data items are available with both normal items and anomalous items [47, 74]. One major challenge for anomaly detection learning in this category is the ratio of available

normal data instances to non-normal data instances, especially considering that anomalous instances are often comparatively rare events [23, 74]. Given a large number of anomalous data instances, this type of approach often leads to an anomaly detection result with high precision.

Nevertheless, as the approaches under this category have known anomaly status, first, it is challenging to obtain data or simulate anomaly instances that are representative of the real world; second, this type of anomaly detection technique has the shortcoming of not detecting previously unknown anomaly instances [23]. For example, supervised approaches have been reported to be vulnerable against new types of malicious attacks in the domain of network intrusion detection [76]. The approach proposed in this thesis does not fit into this category, as the training data in our approach has no dependency on labelled data.

Unsupervised Anomaly Detection: In this type of technique, data do not require labelled data, and thereby, data have a wider range of applicability. This technique has an underlying assumption that the number of normal data points is more frequent than anomalous data points, otherwise, the anomaly detection may lead to too many false negative alarms [23, 76]. This technique has the advantage of being employed for the environments where changes are usually perceived. This type of anomaly detection approach is an especially good candidate for "detecting unknown anomalies in cloud data centres where precise definition of anomaly characteristics may not always exist" [60]. The

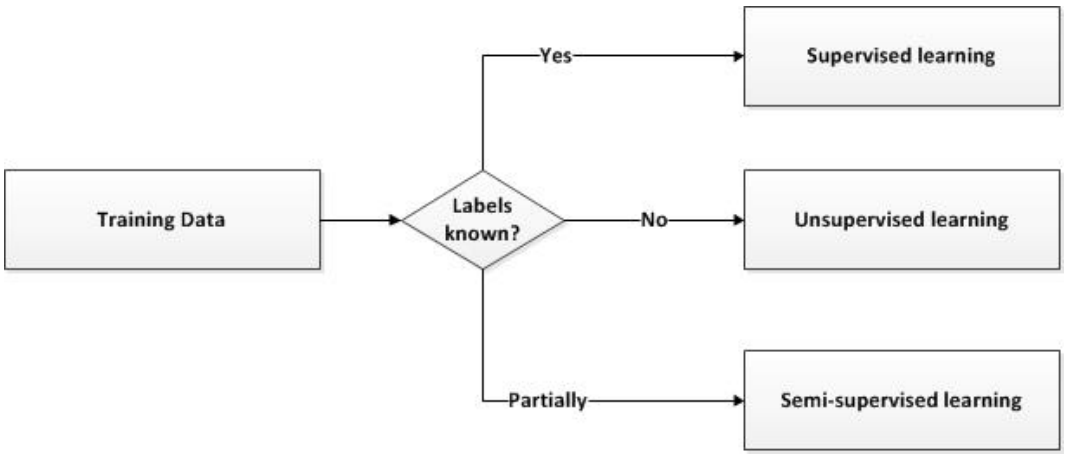


Figure 2.3: Relationship between type of training data and type of anomaly detection technique

approach proposed in this thesis has a similar assumption. In other words, the learning phase of our approach can be conducted with a data set without labelling data when the presence of anomalous instances is far less than normal instances.

Semi-supervised Anomaly Detection: This technique aims to bring the benefits of both the approaches of supervised and unsupervised techniques. Semi-supervised methods have the underlying assumption that labelled data are available for a portion of data, while the rest of the data are unlabelled [60]. Thus, learning from a small chunk of data helps to build a learning structure for the remaining data. Although this technique, unlike the supervised technique, does not require labelling of all data instances, the need for labelling data for a portion of learning data still makes it less applicable for many environments where such labelling data are not available [23].

2.5.3 Anomaly Detection Characterisation: Types of Anomalies

Anomaly detection approaches are broadly categorised into three types: point anomalies, contextual anomalies, and collective anomalies.

Point Anomalies: "A point anomaly is any point that deviates from the range of expected values in a given set of data" [60]. In this type of anomaly,

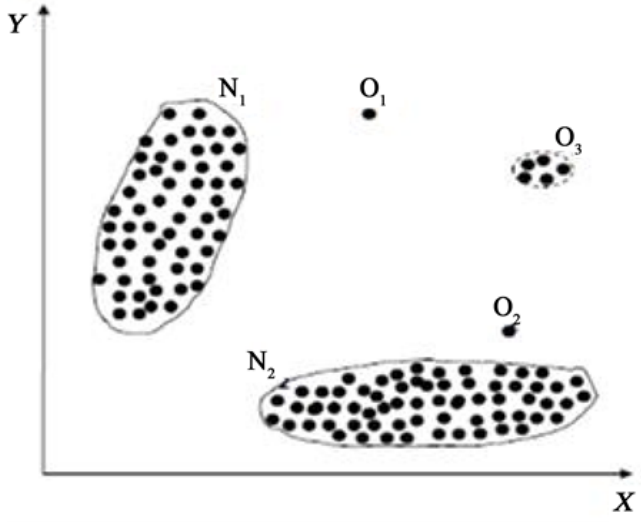


Figure 2.4: An illustration of **point anomaly**: N_1 and N_2 are sets of normal points, while O_{1-3} are outliers or sets of anomalous points.

a distinct point in data is considered anomalous in comparison with the rest of the data, thereby the detected anomalous instance is referred to as a point anomaly. The vast majority of research on anomaly detection has focused on point anomalies [23]. A simple method in this type of anomaly detection technique is to define a statistic rule to identify outliers within a range of data. For instance, whenever the value of a data instance of a metric (e.g. number of open service sessions in memory) deviates more than three standard deviations from the mean, it is considered an outlier, and, as a result, an alarm should be triggered. In system performance monitoring, this type of anomaly detection has been broadly used for detection of spikes in system resource utilisation [60]. An illustration of point anomaly is shown in Fig. 2.4

Contextual Anomalies: This type of anomaly detection technique, also called conditional anomaly, inspects a data instance according to the given context of that data instance. In other words, "These anomalies manifest themselves under specific conditions or contexts" [60]. For instance, high network throughput during working hours is considered normal behaviour, while if such high traffic is perceived after working hours, then that is considered anomalous behaviour. Some common examples of contextual attributes are time or date (e.g. hours of a day, day of a week, or special calendar event), location (e.g. longitude and latitude, IP address), system configurations, and type of workload. An illustration of contextual anomaly is shown in Fig. 2.5.

Collective Anomalies: In this type of anomaly, the detection of anoma-

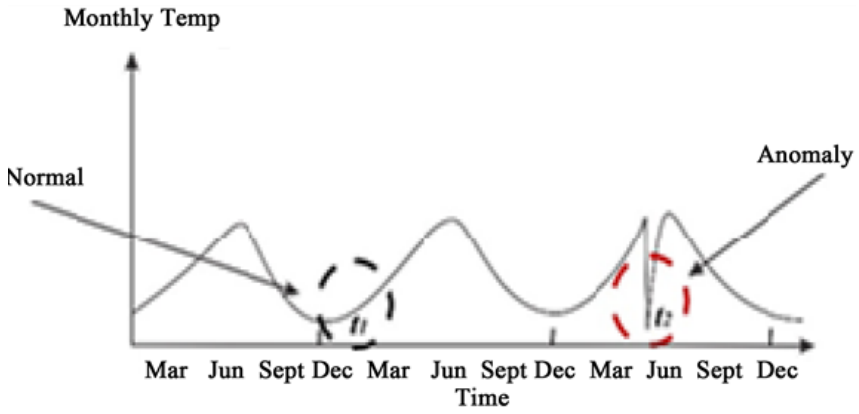


Figure 2.5: An illustration of **contextual anomaly**, month of a year (time) as the contextual attribute for monitoring normality of temperature.

lous incidents is analysed in a group of data instances rather than individual data points. For instance, the anomaly instances are interpreted according to the value of their neighbours, such as an observation of low-throughput compared with higher-throughput value in prior observation windows [60]. Hence, the notion of normal behaviour has a dynamic nature, which may change from one range of data to another (e.g. comparing current value with previous time windows). An emergent subcategory of collective anomalies is **pattern anomalies**. In this technique, anomaly detection is conducted based on detecting anomalous patterns, as metrics may manifest themselves in specific shapes and graphs [4, 52]. This pattern recognition is usually done with the help of mathematical models. The major drawback of pattern-based anomaly detection is the high computational complexity of continuously analysing patterns [4]. An illustration of contextual anomaly is shown in Fig. 2.6.

In this thesis, our anomaly detection approach inspects point data from resource metrics according to what is expected from the context of the activity logs. Therefore, our approach is mainly a contextual based technique.

2.5.4 Challenges in Anomaly Detection

In short, an anomalous behaviour is a deviation from normal behaviour. Therefore, to detect deviations from normal behaviour, a criterion or measurement is needed to define normal behaviour. However, this seemingly simple task of defining a range or criterion for normal behaviour and distinguishing it from abnormal behaviour is very challenging in practice. Chandola and his colleagues [23] outline these challenges as the following:

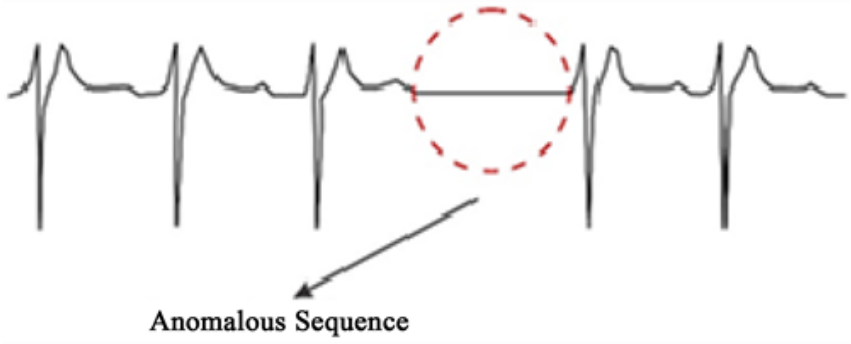


Figure 2.6: An illustration of **collective anomaly**

- **Defining normal region:** Defining proper criteria that indicate the range of normal data instances is often a difficult task. This is especially challenging when we need to draw a line between normal data instances and anomalous data instances. In fact, in many cases, it is likely that data in a normal region that is close to a borderline is an indicator of an anomaly, and vice versa.

- **Detecting malicious actions:** Malicious actions may adapt themselves to mimic normal behaviour in a system, thereby making it difficult to distinguish normal behaviour from non-normal behaviour.

- **Evolution of normal behaviour:** If the characteristics of the current notion of normal behaviour change, then the normal behaviour needs to be updated as normal behaviour changes.

- **Different domains require different anomaly notions:** The exact notion of an anomaly varies from one application domain to another. For example, while fluctuations in the value of a stock market may be accounted as normal, in the medical field, a minimal deviation from the normal value of a blood test can be considered anomalous. This makes the adaptation of an effective anomaly detection technique of one domain not suitable in another domain.

- **Lack of labelled data**: Many anomaly detection techniques rely on labelled data, while for much training data, such labelling is not available.

- **Distinguishing noise from anomalies:** There is a chance that noise is similar to anomalous data instances, thereby making a differentiation between these two a challenging task.

The challenges outlined above make the detection of anomalies a challenging task. As a result, most anomaly detection techniques are designed for a particular domain, and choosing a proper anomaly detection method requires consideration of a set of factors such as the field of work, the characteristics of the available data, the type of anomalies to be detected, and data dependency and data volume [73]. Thus, in this chapter, we review the anomaly detection

techniques that have been adopted, to some extent, in the domain of system health and performance monitoring, respectively.

2.6 Anomaly Detection Techniques

In the previous subsection, we discussed the three categories of anomaly detection approaches. In this section, we will present the classification of different anomaly detection techniques and discuss the strengths and drawbacks of each type of technique.

The domain of failure and anomaly detection is very broad, and few studies have attempted to provide a structural survey of general anomaly detection techniques and applications. A comprehensive systematic analysis of anomaly detection techniques and their applications in different domains has been given in [23]. Further extensive analyses and surveys on anomaly detection methods can be found in [3, 11, 56, 60, 103]. The focus of the current thesis is on system health, operation, and performance monitoring. Thus, we focus on this domain for analysis of the related work.

2.6.1 Classification-based Techniques

Classification methods are a type of supervised machine learning method which requires labelled data. In this technique, the data instances with normal and anomalous labels are used to learn a model called a classifier. Then the classifier from the training phase is used to test the accuracy of the classifier to distinguish normal data instances from anomalous ones [104]. There are several classification techniques available, but they can be categorised into two general groups of one-class and multi-class anomaly detection techniques [23]. One-class anomaly detection techniques assume data has one type of normal instance, while for the case of multi-class anomaly detection methods, there are multiple types of normal instances in a dataset. Some popular classification methods include Bayesian networks, neural networks, and decision trees. [60].

Related to these types of techniques, Tan et al. [115] attempted to predict

and classify performance anomalies for virtualised cloud computing infrastructure using a Bayesian classifier and tree augmented network. Classification as a decision tree has been leveraged by [44, 101] to detect performance bottlenecks and degradation. Also, Parekh et al. [101] and Powers et al. [107] present a comparative and exploratory study of several classification techniques for performance anomaly detection.

The process complexity of the classification methods varies widely from one classification algorithm to another, nevertheless, for most of the classification algorithms, the testing phase is fast in comparison to other approaches [60]. Unlike the adopted technique in this thesis, classification techniques require anomalous labelled data instances, which is often difficult to obtain in the domain of performance anomaly detection. Further, this type of technique has weaknesses as regards detecting the unknown type of anomalies.

2.6.2 Clustering-based Techniques

Clustering approaches rely on grouping similar data instances into separate clusters. Clustering anomaly detection techniques work based on one of the following assumptions: normal data instances belong to a cluster, while anomalies do not belong to any cluster; anomalous instances are data instances that are far away from a cluster centroid in oppose to normal data instances; normal data instances belong to large and dense clusters in compare to anomalous data instances that belong to smaller or spare clusters [23].

In contrast to classification-based methods, clustering techniques are often unsupervised machine learning methods, though semi-supervised methods also have been proposed recently [100]. In clustering techniques, outlier detection relies on measuring the distance of data, often based on one or a combination of the following rules. Data instances are anomalous if:

- (i) they do not belong to any cluster, or
- (ii) they are far away from their cluster centre, or
- (iii) they belong to sparse clusters [23].

Some of the popular clustering methods are self-organization maps clustering, K-means clustering, and expectation maximization [56, 60].

Related to this technique in the domain of system anomaly detection, Dean et al. present an unsupervised clustering method using self-organizing maps to distinguish the normal states of the system from the unhealthy states, and then use a decision tree mechanism to detect failures in the cloud environment [33]. Also, Yu and Lan [142] present an approach using majority-voting, a non-parametric clustering technique, for anomaly detection in Hadoop clusters.

Clustering methods have the advantage of working in an unsupervised mode (similar to this thesis), and they are often reported to be fast in the testing phase, as the number of clusters is usually small [23]. As for the weakness of clustering techniques, computational complexity at the learning phase is often expensive, and if the number and association of anomalous instances are not significant enough to form clusters, then a clustering algorithm will not be effective [23]. Another difficulty with this approach is the processing of complex data, such as streaming and spatial data, for which creating a distance measure is often complicated [60].

2.6.3 Nearest Neighbour-based Techniques

Nearest neighbour-based techniques are unsupervised techniques and they have some similarities to clustering-based techniques. Like clustering techniques, in nearest neighbour-based techniques, the distance computation between pair instances is required for classification purposes. However, rather than the distance of instances to be compared with the centre of the clusters, the distances are analysed with respect to their dense neighbourhood. The base hypothesis in this type of technique is that normal data instances are located in dense neighbourhoods, while anomalous data instances tend to be located at some distance from their closest neighbours [23]. Two of the popular techniques of this type are k-nearest neighbour [66] and the local outlier factor [105].

An example of this technique is the study proposed by Huang et al. that uses a local outlier factor method to detect contextual anomalies in cloud computing systems [58]. Similarly, Wang et al. introduce an algorithm that

learns from workload patterns in a web application and then adopts the local outlier factor for anomaly detection of such patterns [127]. Further, Bhaduri et al. leverage a distance-based anomaly detection rule using the K-nearest neighbour method to detect machine failures in cloud infrastructure [14]. The advantages and drawbacks of nearest neighbour-based techniques are often similar to clustering methods.

2.6.4 Information Theoretic Techniques

This category is introduced for the first time in [23] and includes approaches that find anomalies based on irregularities in the content of a dataset. Irregularities, or the degree of dispersal and concentration of content in a data set [126], is measured using techniques such as Kolmogorov Complexity [124], entropy, relative uncertainty [75] and similar techniques [23].

The above techniques in system monitoring can be applied to detect anomalies by measuring the differences between two windows of metric observation [60]. In this process, an entropy-based technique has been adopted by [125] and [75] to identify malicious activities in network traffic. Wang et al. [126] used a similar approach for detecting anomalies in cloud environments. This technique has the advantage of working with unsupervised data and has no restriction on the data type distribution. However, finding an optimal threshold for the size of the substructure of data is challenging. Further, in contrast to the approach adopted in this thesis, this approach is more efficient when there is a significant number of anomalous instances available [23].

2.6.5 Statistical Techniques

Techniques in this category utilise various statistical-based methods to analyse the probability distribution of data and the relationship between them to construct a model that defines the range of data in normal states. Broadly speaking, statistical techniques can be divided into two groups: *parametric* and *non-parametric* techniques.

Parametric Methods

Three broadly used parametric methods are the Gaussian-based anomaly detection technique, the correlation-based anomaly detection technique, and the regression-based anomaly detection technique.

The Gaussian-based anomaly detection technique is one of the most utilised methods in the literature, which has the underlying assumption that data distribution is normal. Mean and variance are two parameters that are used to build this type of model, and the distance of a data instance from the mean is used as a threshold to detect anomalous items [23]. For example, a data instance that is not within three standard deviations from the mean is considered to be an outlier.

The correlation-based technique is another type of parametric statistical technique that measures the interdependency between two sets of data. For instance, this method is used to find the correlation between two different metrics or two subsets of data of the same metric. In this method, *coefficient R* is calculated by various algorithms, such as the *Pearson* or *Spearsman* correlation, to measure the association of two variables. In this technique, the correlation of two sets of data, often within a time window, is measured, and if it is not between the expected minimum and the maximum coefficient R, then data records will be tagged as anomalous data instances. In this thesis, we have extensively leveraged the Pearson coefficient correlation for finding associations between log events but not for the sake of anomaly detection.

The regression-based technique is another method that has been used for system performance anomaly detection, especially for time-series data. In regression analysis, there are two types of variables for statistical analysis: response variables, also named dependent variables (DV) [89], and predictor variables, also called independent variables (IV). The *Dependent Variable* is a predicted variable in response to changes in predictors. In other words, the dependent variable is the response value that, with the help of the regression model, is desired to be explained by the predictors. The *Independent Variable* predicts the value of the dependent variable. In other words, regression is used to explain the variation in a DV by analysis of the variation of IVs. Thus,

IVs are the variables that an experimenter may manipulate in order to derive a statistical model that has the best prediction ability. In short, regression technique can be used to understand how much of the variation in a DV or target variable can be explained by a set of IVs or explanatory variables.

The objective of the regression-based method is to fit a regression model that has the least amount of absolute error [60, 69]. Once the model is fitted, the computed residuals (the error of estimate) are used to detect anomalous data instances which are not within the range of the residual score of the estimate. In this thesis, we adopt the regression-based technique for our anomaly detection approach.

Non-parametric Methods

Several other statistical methods have been used for anomaly detection in the past. Nonparametric methods are a subset of these methods which build a profile of normal data from the given data using methods such as a histogram. Histograms are used as one of the simplest non-parametric statistical techniques to maintain a pattern profile of normal data. Histogram-based techniques are particularly popular in the domain of network intrusion detection and fraud detection [23].

In the process of anomaly detection with a histogram for univariate data, first a histogram is built based on values from the training data, then the testing data will be checked to see if they fall in the bins of the histogram. The data that do not fall in the histogram bins will be tagged as anomalous data. In this method, "the size of the bin used when building the histogram is key for anomaly detection: if the bins are small, many normal test instances will fall in empty or rare bins, resulting in a high false alarm rate; if the bins are large, many anomalous test instances will fall in frequent bins, resulting in a high false negative rate" [23]. Therefore, one of the challenges in the histogram-based technique is to define the optimal size of bins, in order to get the best anomaly detection result.

Non-parametric methods are often simpler than parametric methods to implement when dealing with univariate data, but when we need to work with

multivariate data, they are not able to capture the interaction between different attributes. This limitation makes this technique not suitable for detecting anomalies that are only perceived as the combination of multiple variables, such as the work of this thesis, which takes into account multiple independent variables derived from event logs to predict the status of a resource metric.

The models derived from statistical anomaly detection techniques often offer robust solutions, as they are based on statistical significance, though applying statistical techniques has the challenge of finding the best statistical model to fit the data. Anomaly detection based on statistical techniques has the advantage that the detection of anomalous instances often comes with a confidence interval derived from the anomaly score [23]. This confidence indicator can be used as additional information for decision making, such as for the prioritising of anomaly alarms. In addition, statistical techniques often have the benefit of being employed for unsupervised data instances when the number of anomalous data instances is far less than the normal data instances [23].

2.7 Summary

In this chapter, we gave a background on the importance of system monitoring, especially for the environment with virtualization technologies. We also highlighted the importance of DevOps operations in this domain. Moreover, we provided a background on common anomaly detection techniques and summarised the advantages and disadvantages of each technique.

Please note that the closely related works of this thesis are discussed after presenting and discussing the proposed approach of the thesis in Chapter 3, Section 3.5.

Chapter 3

Metric Selection and Unsupervised Anomaly Detection Using Log-Metric Regression Analysis

This chapter provides a conceptual framework and presents a structured view of the approach that is proposed in this thesis. First, the overview of the steps of the research approach is outlined, then the mechanisms that are employed for each of these steps are explained and elaborated. In this direction, in this chapter we will explain and discuss the process of data collection and metric derivation. Next, we explain the log abstraction process and log-metric mapping. Followed by a section about how to find correlation between log events and resource metrics. We then use the correlation model derived in the previous step for metric selection and later for assertion specification and anomaly detection.[1]

3.1 Overview of the Proposed Approach

The goal of the approach of this thesis is to address the challenges and limitations outlined in Section 1.1 and to fulfil the objective of the thesis highlighted in Section 1.2. In particular, the approach in this chapter proposes a metric selection and anomaly detection technique for system health and performance

[1]Parts of this chapter have been published: [36, 37].

monitoring. Unlike many anomaly detection methods, the proposed approach is an unsupervised approach, which does not require anomalous labelled data [47, 74]. Also, it proposes a context-based anomaly detection technique. The context-based technique (also referred to as the behavioural or conditional anomaly detection technique) means observed data instances are considered anomalous in association with behavioural attributes of a system [23].

Logs represent the behaviour (context) of system operation, while metrics show the health, system, and performance status of a system. Several studies have leveraged logs for failure detection and diagnosis, and a large number of studies have adopted metrics for system health and performance anomaly detection e.g. [23, 60, 143]. Some of these studies are discussed in Chapter 2. However, the combination of these two valuable sources of information in an integrated form for system monitoring and anomaly detection has not been studied before.

Combining these two sources provides the benefits to identify the anomalies that are not detected in either of these sources of data alone. In this direction, we propose and employ a set of techniques to combine information from logs and metrics based on statistical analysis without requirement of domain knowledge of the system.

Event logs are textual content, while system resource metrics are often represented with numerical values. As can be seen in Fig. 3.1 and Fig. 3.2, event logs are generated at various points in time, and metrics are collected at potentially different points in time.

- ***Events*** refer to individual log lines. We assume each log line has a timestamp and a description.
- ***Clustering*** is a process that clusters fine-grained correlated event logs to set of coarse-grained activities. We refer to this process as log abstraction or log clustering.
- ***Activities*** refer to sets of log events that together are responsible for making a change in a system or indicating a status of a system or application, for instance, retrieving user profile information or launching a VM instance.

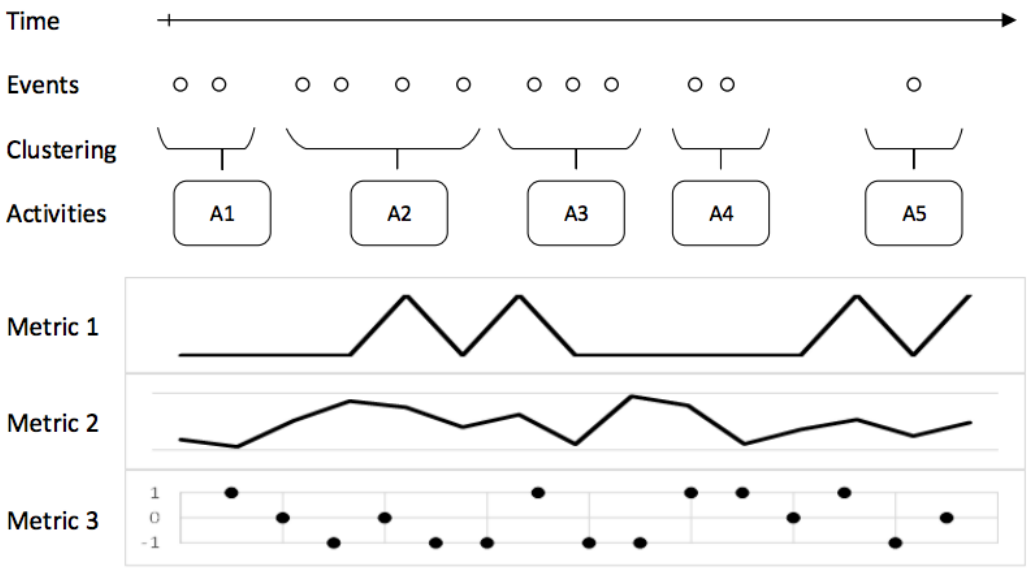


Figure 3.1: The occurrence of log events and metric data over time. *Note: A1..A5 are activities; Metric1..3 are metrics like CPU utilisation, network usage, number of instances changes, etc.

- ***Metrics*** represent status and utilisation of system resources within a time window, for instance, average CPU utilisation or network usage within the last 5 seconds.

Detailed system monitoring is often a cost-intensive process, and the monitoring metrics are often collected with longer frequency than logs [20, 67, 91]. Moreover, collecting monitoring data usually happens at fixed intervals (such as every 5 seconds, 1 minute, or every 5 minutes). In contrast, observation of system operations behaviour through event logs happens at non-fixed intervals, such as the occurrence of a few event logs within one second, and then the absence of any event logs for the next few seconds or even minutes. For example, Fig. 3.2 shows a sample of the logs of a JBoss application server. By looking at the timestamps, we can observe that logs are happening with different time intervals. Also, in this figure we show a few examples of grouping a set of related log events to a set of activities. We will explain the log abstraction process in Section 5.3.

In our investigation, we are interested to find the relationship between operation activities derived from log events and the symptoms observed in resource metrics. Once we map logs and metrics, we adopt a statistically justified tech-

nique using correlation analysis to narrow down the number of metrics to the most sensitive metrics influenced by application operation behaviour that is reflected in event logs. Then, our approach uses a statistical technique to extract a regression-based model that explains the correlation and potential causalities between operation event logs and resource metrics. 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 metric that can be observed. The output of the regression model is used to generate assertions, which are then leveraged for anomaly detection of run-time execution of application operations. The high-level steps of the approach, also shown in Fig. 3.3, are as follows:

1. data collection and data preparation;
2. metrics derivation;
3. log abstraction and log-metrics data mapping;
4. correlation derivation between logs and metrics;
5. metric selection; and
6. assertion specification for anomaly detection.

We will describe and discuss above steps in detail in next sections.

Figure 3.2: An example of JBoss logs with clustering related log events to set of activities.

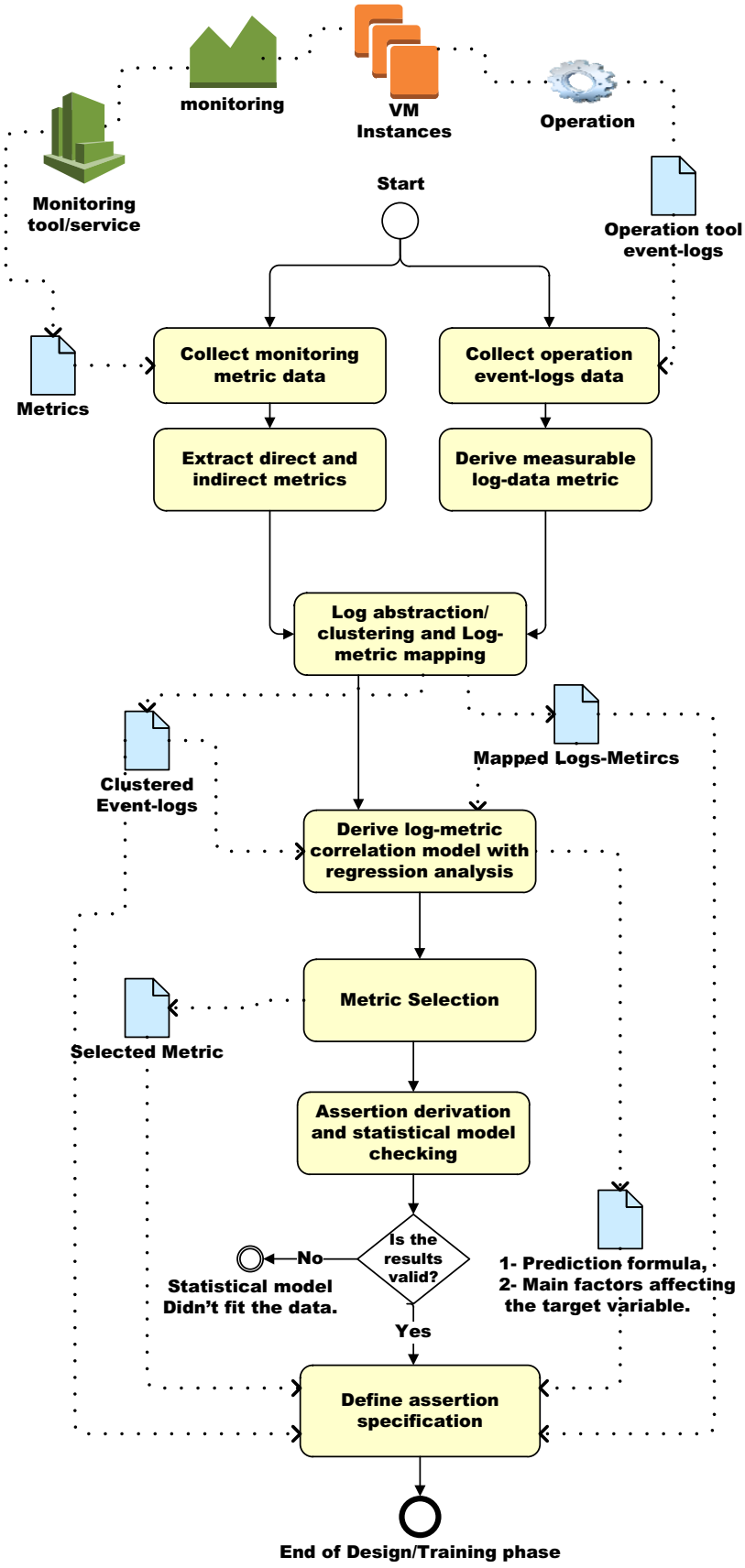


Figure 3.3: Workflow of the proposed framework.

3.2 Sources of Monitoring Data and Data Preparation

We describe the steps of data collection for both monitoring metrics and logs in this section.

3.2.1 Event Logs from Operation Tools

While the status of resources is often monitored through metrics, logs are the main source of information that reports what functions are executed in a system [58]. Log events explain the behaviour of the applications and operations, and, in our context-based anomaly detection approach, is the main source to extract the events of a system [58, 96].

In the approach presented in this thesis, to draw mappings between numerical metrics and contextual 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 types of log contain time-stamped information, whether they are application logs, database logs, or operation logs. Furthermore, a log message represents information about an event, including logs that indicate preparation or waiting periods.

The timestamp and description attributes can be seen in almost all operational logs, though the quality and granularity of logs may differ from one application to another. For instance, middleware application servers and operational tools such as Apache Tomcat[2], Eclipse Jetty[3],JBoss [4], Asgard[5], and CloudTrail[6] provide logs that contain these types of information (logs with timestamp and event description).

Hence, this study assumes a logged event must have at least two attributes: a *timestamp* and an *event description*. Knowledge derived from logs can be leveraged to monitor the behaviour of a system and detect errors and diagnosis the root cause failure [28, 29]. In the approach proposed in this thesis, we use

[2]https://tomcat.apache.org/
[3]http://www.eclipse.org/jetty/
[4]http://www.jboss.org/technology/
[5]Netflix Deployment Tool - https://github.com/Netflix/asgard/wiki
[6]Logs reporting service of Amazon AWS - https://aws.amazon.com/cloudtrail/

the information from logs, along with metrics, to identify anomalies in a system operation.

3.2.2 Metrics from Monitoring Tools

The approach of this study requires two sources of monitoring data: one is the set of log events of running operations and the other is the metrics from the resources. Both of these types of monitoring data are available in almost all enterprise application systems. For example, public and private cloud service providers offer monitoring services, namely: Azure Monitor for Microsoft Azure[7], Google Stackdriver for Google cloud[8], CloudWatch for Amazon AWS [9], LogicMonitor [10] and many other third-party services and application performance monitoring solution providers[11].

Monitoring tools provide multiple metrics to show the status of various system and application resources. For example, at the operating system level, one can collect monitoring data of running processes, system activities and hardware and system information either using native Linux tools like "top" and "vmstat" or utilising more sophisticated monitoring solutions like Nagios.[12] As another example, public cloud service providers such as Microsoft Azure and Google Cloud provide several monitoring metrics which are available at both individual resource level (e.g. storage, or VM instances level) and at a group level, showing the aggregated form of multiple resources (e.g. the average disk write usage for 10 storage devices).

Typically, system monitoring can be conducted in two forms: through performance-based metrics, or state-based metrics.

- **Performance or direct metrics**: These basic metrics are often available by default with the monitoring tools and available for most of the metrics that show the capacity or performance of various resources.

[7] http://azure.microsoft.com/
[8] http://cloud.google.com/stackdriver
[9] http://aws.amazon.com/cloudwatch/
[10] http://www.logicmonitor.com/
[11] http://haydenjames.io/20-top-server-monitoring-application-performance-monitoring-apm-solutions/
[12] https://www.nagios.com/

These types of metrics are the metrics that show the percentage or the degree of the changes of the usage of the resources, such as CPU utilisation, disk read IO, memory usage, network traffic, number of SQL threads running, number of open connections, etc. A list of hundreds of these metrics from Google cloud platform metrics, Stackdrive Monitoring Agent metrics and Amazon Web Services metrics can be found in the link in footnote [13].

- **State-based or derived metrics**: These metrics may not always be available directly and they often indicate the transition of the status of resources from one state to another. One important example of these types of metrics are the ones indicating the status of a VM. For example CloudWatch (AWS monitoring service), includes no metric that explicitly shows the number of instances started or terminated within an auto scaling group. Fig. 3.4 shows a typical state flow of the lifecycle of a VM [55].

As shown in Fig. 3.4:

- **Uninstantiated** is the pre-creation or baseline state of a VM and may involve platform-dependent operations. For instance, a pre-state of VM creation is baking an Amazon Machine Image (AMI) which has the configuration of creating a virtual machine.
- **Waiting** often refers to the state of resource allocation and booting process.
- **Running** is the state of a VM that is ready to provide services.
- **Paused** refers to a temporary suspension of a VM which is stored in the memory.
- **Suspended** is also called "stopped" and indicates a suspension of a VM but the state of the VM is persisted on the physical disk rather than the memory.

[13] https://cloud.google.com/monitoring/api/metrics

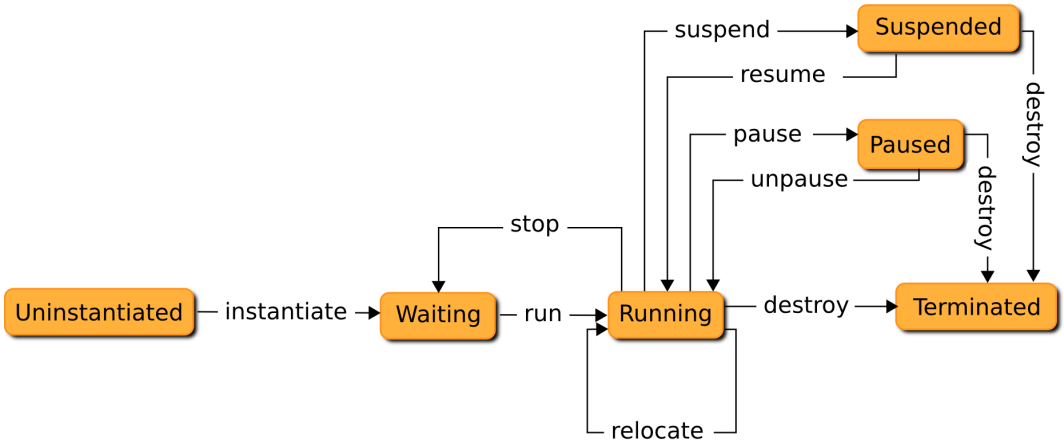


Figure 3.4: Sample lifecycle of a virtual machine with transient actions and states from one state to another.

- **Terminated** is the state in which a VM is destroyed and thus cannot be restored.

In this thesis, we will explore both types of performance metrics (direct metrics) and state-based metrics (derived metrics) and evaluate the applicability and effectiveness of our approach for each of these groups.

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 Section 3.4.1, we will demonstrate how to leverage a statistical supported technique to identify the most sensitive metrics for detection of anomalies in a system.

3.3 Log Processing and Log Abstraction

3.3.1 Representing Logs as Quantitative Metrics

Raw log events are often unstructured, as developers write log messages as free text for convenience and flexibility [15]. Also, system logs are often available in large numbers and they are fine-grained [43, 96]. This makes the monitoring and troubleshooting of systems using raw logs a cumbersome task, especially in large-scale computing environments. For instance, the number of registered log events in an application server such as Tomcat or JBoss can reach over

dozens or hundreds of log lines per minute. Nevertheless, most of the log events are often recurring, indicating occurrences of similar events.

For our study, to observe how the activities reported on log events change the state of resources, we are interested in tracking reoccurring log events. So we need a way to parse and trace log events. A typical approach for log parsing is to utilise regular expression to derive a template of event logs [54, 114, 117, 130, 144]. Similarly, we employed the common approach of using regular expressions to identify log event types. In this process, the raw log messages are transformed into a set of structured log events.

As shown in the example in Fig. 3.2, log messages typically come with a timestamp and message content. The log message describes what is happening at runtime [114]. Thus, for the log processing in our approach, we assume logs contain both timestamps and a message description. The message part of a log event of a particular type can be divided into two parts: the constant part and the variable part [54]. The constant part of the message is fixed and remains the same for every log event of this type. The variable part holds the runtime information of log events like an IP address and port number [54].

The goal of log parsing in our approach is to extract the pattern of recurring event logs by automatically separating the constant parts and variable parts of a raw log message, and further transform each log message associate with a specific log event type. To fulfil this aim, we process log files through the following steps.

First, the timestamps and log description are extracted for each log line, followed by tokenisation of the log message. Next, regular expressions are generated for each token of the log message. Then, message tokens are divided into two parts: constant tokens and variable tokens, by analysing the pattern of regular expressions, using a set of pre-defined rules. Lastly, the generated regular expressions are combined to form a unique log event type. For each new log line, the log event is compared by pattern matching with regular expressions, and if a pattern is not found, the above steps will be repeated for the new log line.

The rules we used in our log analysis with regular expressions include com-

mon patterns of URL address, IP address, port number, camel-case strings, and numerical values (which may include time duration, IDs, etc.). This method is simple and shown to be effective in past studies [117, 138, 140], however, these rules may need to be configured or extended for parsing logs of a new project.

In the process of log transformation, the variable tokens are replaced with the regular expression for the token type. For instance, string token *process[11043]* is replaced by *process[\d+]*, "\d+" indicates that the string must have one or more numbers as a variable in the brackets; the rest of the token is taken as an exact expression. a complete example of log transformation is shown below - "\s" is an indicator of space:

```
``Remove instances [i-08b43bc7] from Load Balancer ELB-01''
```

generate the following regular expression:

```
``Remove\sinstances\s\[i-[a-z0-9]+\]\sfrom\sLoad\sBalancer\s.*''
```

The output of the above step is a set of regular expression patterns that represent unique types of event logs.

Interpolate Occurrence Strength of Log Types

Once we have parsed the logs and identified what type each log event belongs to, we can start counting the occurrence of each log event type, however, we need a mechanism to map the counting extracted from the event logs to the time window for which our system health and performance metrics are available. For instance, if metrics are available with one-minute intervals, then to model the relationship between logs and metrics, we need to represent the metrics extracted from the logs with the same time window.

To the best of our knowledge, this matter has not been explored in previous approaches for log analysis, perhaps as most of these past approaches had a sole focus on logs, rather than considering other factors like metrics. Also, unlike many past studies [18, 43, 114, 119, 134] that employ feature extractions in order to track what happens in a log, in our approach, we propose a simple method that does not have an interest in interpreting the context of the log message.

In fact, the focus of this thesis is to find out whether the mere occurrence of individual log events, without having domain knowledge of the context of the logs, can be employed alongside resource metrics in order to improve metric selection and anomaly detection. In this direction, and since many system resource monitoring (such as in the case of public cloud services) there are limited options available for configuring the frequency of resource monitoring, we propose a weight-timing method that performs counting based on the *interpolated occurrence strength* of each log event within a time window.

It is worth noting that, in the beginning, we started our investigation based on the simple counting of occurrence of log event types without interpolated values, but this didn't lead to sensible clustering due to the collinearity observed among event types in our experiments, therefore, we incorporated interpolated occurrence strength into our analysis.

In this method, the *interpolation* indicates at which unit (e.g. second) of a time window (e.g. minute) an event happened. Indicating a point of time for the derived metric for log events would show a relative interval of the occurrence of a set of log events. To this end, we parse the timestamp of each log message and extract the point of time (e.g. 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 time window (e.g. minute) and a time window before and after the event happened.

Given a log event type is denoted as e, the smallest unit of time that logs can track is denoted as x, the interval of time that monitoring metrics are available is denoted as tw, and D_{tw} represents the duration of time window, then the weight-timing occurrence of an event type at time x of a full time window can be obtained:

$$e_{n(tw)} = \frac{D_{tw} - x}{D_{tw}}$$

The weight-timing for the interpolated value of the above event for the next minute can be obtained from:

$$e_{n(tw+1)} = \frac{x}{D_{tw}}$$

So for the sum of n times occurrences of an event type for a time window, we have:

$$E_{tw} = \sum_{i=1}^{i=n} e_{n(tw)}$$

To give an example, let us assume we have a time window of 1 minute (60 seconds) duration, and event $e1$ happened once at 25 seconds, the second time it happens at 40 seconds, and suppose there is no occurrence of e1 in the following minutes, then its occurrence strength for the current minute and the next minute for the first and second log occurrences are obtained as:

For the first occurrence of event type:

$$e1_{tw} = \frac{60 - 25}{60} = 0.583 \qquad e1_{tw+1} = \frac{25}{60} = 0.417$$

$$e1_{tw} = \frac{60 - 40}{60} = 0.333 \qquad e1_{tw+1} = \frac{40}{60} = 0.666$$

Then the interpolated occurrence strength for e1 in the current time window and the next time window are obtained as follows:

$$E1_{tw} = 0.588 + 0.333 = 0.921$$

$$E1_{tw+1} = 0.417 + 0.666 = 1.083$$

The outcome of the above step is a matrix of interpolated occurrence strength, as shown in Table 3.1), for each event type at each time window. This outcome allows us to employ statistical analysis and find the correlation between log event types and clustering log events to a set of activities. Also, with this approach we are able to map the event logs into the time window interval of the monitoring resource metrics of our system.

Table 3.1: Matrix of interpolated occurrence strength for each event type.

Timestamp	E1	E2	....	Ek
$tw1$	$E1_{tw1}$	$E2_{tw1}$	...	Ek_{tw1}
$tw2$	$E1_{tw2}$	$E2_{tw2}$	...	Ek_{tw2}
$tw3$	$E1_{tw3}$	$E2_{tw3}$	...	Ek_{tw3}
...	...	...	...	...
...	...	...	...	...
twn	$E1_{twn}$	$E2_{twn}$	...	Ek_{twn}

3.3.2 Abstracting Event Logs to Activities

Systems application and operations logs are often more fine-grained than the activities that are at the business level or the level that affects the status of resources. The increasing volume and complexity of logs have made the monitoring of system behaviour through logs a very daunting task [58]. Hence, many application operation tools, such as JBoss or Tomcat provide a mode of producing logs at either a verbose or a non-verbose logging level. In many cases, even at a non-verbose level, it is often a set of log events together that report an execution of a use case. For example, terminating a VM instance from an Auto Scaling Group in Amazon EC2 leaves the trace of five unique log events in an operation tool like CloudTrail or Asgard. This issue has motivated the emergence of log abstraction techniques [43, 63, 82, 92]. In fact, many log file analysis methods rely on log abstraction, such as for failure troubleshooting [84], operational profiling [93], and anomaly detection [43, 136].

Given the above matter and the objective of this study, we abstracted logged events into higher-level activities for the following reasons:

- Most often a system behaviour is not characterised by a single log event; typically, a set of log events together cause a tangible impact on the status of resources, thus, to find the impact of event logs on resources we need to cluster related log events.

- If a set of event types always co-occur, then high correlation among them may cause a problem of multicollinearity in some statistical models,

which can lead to unreliable and unstable estimates.

- Logs are often low-level and voluminous; by raising the level of abstraction, the tracking of system activities becomes simplified, and users may find the information provided more useful [10, 58].

In order to find the related log events, we adopted the Pearson product-moment correlation coefficient. The Pearson product-moment correlation coefficient is a measure of the strength of the linear relationship between two variables [98], thus, we have adopted this method to derive a measure of association strength between two pairs of event logs.

The Pearson correlation coefficient is commonly represented by the symbol r. Given we have one dataset $x_1, ..., x_n$ containing n values and another dataset $y_1, ..., y_n$ containing n values then r is obtained as follows:

$$r = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n\sum x_i^2 - (\sum x_i)^2][n\sum y_i^2 - (\sum y_i)^2]}}$$

In the case of calculating the correlation strength between two log event types, in the above equation:

- n is the number of monitoring observations
- x_i denotes the *interpolated occurrence strength* of event type x (Section 3.3.1), at time i,
- y_i denotes the interpolated occurrence strength of event type y (Section 3.3.1), at time i.

The value of r obtained from the above equation 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, which, in our case of event log correlation, indicates that events of this types (almost) always co-occur. Negative values indicate that events of the respective event types rarely co-occur. Fig. 3.5 shows a few examples of different values of r in scatter plots.

Therefore, based on the interpolated occurrence strength described in Section 3.3.1, and with the help of the Pearson correlation coefficient, we can

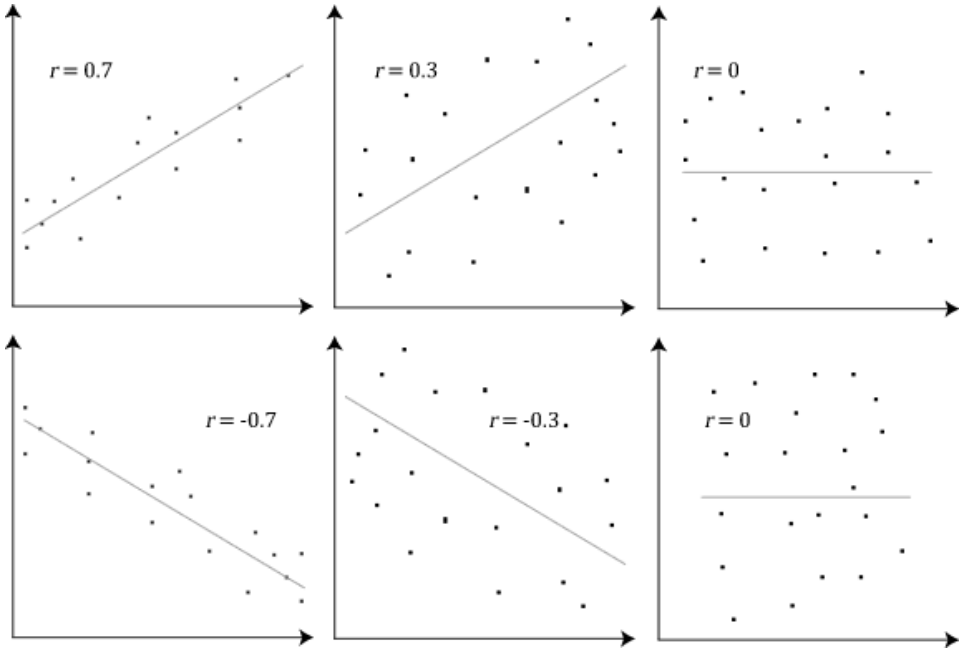


Figure 3.5: Sample of Pearson correlation coefficient output in scatter plots - adopted from [72]

determine where strong associations exist between any two event types. Event types with a very high correlation can then be combined into an activity.

The above proposed log abstraction technique using extracted interpolated occurrence strength is a novel approach in the domain of log abstraction, though we do not claim that the proposed approach outperforms past techniques. To the best of our knowledge, the existing log abstraction techniques [10, 18, 43, 54, 58, 63, 82, 92, 118, 134] do not take into account the fixed time-interval (which is enforced by metric availability, as discussed in Sections 3.3.1) and interpolation occurrences of log event types, whilst our approach effectively addresses this requirement.

Moreover, unlike many log abstraction techniques that analyse the context of logs by using pattern signature and feature extraction methods [18, 43, 114, 119, 134], our approach does not rely on feature extraction from log messages, and does not aim to interpret the context of the log messages. This feature brings the advantage of being less dependent on the quality of the text of the logs, especially as some studies have highlighted significant differences in the quality of different log files [29, 104]. Besides, in contrast to some of the previous studies that required the source code that generated log events

[134–136], another advantage of our approach is that it does not depend on having access to the source code of generated logs. And lastly, our approach is a non-intrusive method, meaning it does not require manipulation of log messages.

3.4 Log-Metric Regression-based Model

For the purpose of this study, we are interested in finding out the effect of operation actions on changes in the status of resources. 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 the correlation between logged events and changes in metrics, that is, the absence, presence, and strength of such changes.

For instance, in a web application where a user signs into a system, a new web session will be created and dedicated to the user by a middleware container like Jetty [14]. Having the assumption that opening a new session is a resource consuming process, then analysing the relationship between event logs that shows users creating a new session by signing into the system, and the observation of system memory, we should be able to learn how much memory is consumed when a new session is created, from the changing values of the memory metric. Such observations can be used to learn about the normal behaviour of the system, and outcome like above can be leveraged for anomaly detection.

Based on the above intuition, we had the hypothesis that the status of system resources can be predicted from activities in the event logs. Each prediction implies that the change in one variable or set of variables together will produce a change in another. Therefore, we have two types of variables for our statistical analysis: response variables, also named dependent variables (DV) [89], and predictor variables, also called independent variables (IV).

- **Dependent Variable**: is the predicted variable in response to changes in the predictor (input) variables. In other words, the dependent vari-

[14] http://www.eclipse.org/jetty/

able is the response value we wish to explain by our predictors. In our approach, this refers to a resource metric.

- **Independent Variable**: predicts the value of the dependent variable. In other words, the variation in a DV is our target, to be explained by the variation of IVs. Thus, IVs are the variables that an experimenter may manipulate in order to derive a statistical model that has the best prediction ability. In our approach, they are the activities abstracted from event logs.

The central idea behind correlation is that two variables have a systematic relationship between their values. This relationship can be positive or negative and varies in strength. We employed the Pearson Coefficient in our previous step to find the correlation between log events. In this stage, we are interested in the relationship between log events and resource metrics. We are not just interested in the correlation between them; we also want to be able to *predict* the values of resource metrics from the activities of log events, as well as wanting to *explain* which activities in a log cause changes to the resources. Therefore, we employ a multiple regression technique to address these needs.

"There are two general applications for multiple regression: prediction and explanation" [99]. This means, first, multiple regression can be utilised to predict an outcome for a particular phenomenon [89]. Second, multiple regression can be used to understand how much of the variation of the outcome can be explained by the correlated variables. Therefore, to understand the relationship between event logs and resource metrics, and to have a predictive model we adopt a multiple regression technique, namely, *Ordinary Least Squares* (OLS) regression [89].

Multiple linear regression attempts to model the relationship between one or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. Given we have one dependent variable y and one independent variable x, the simple regression model will be:

$$y_i = \alpha + \beta x_i + \epsilon_i$$

In this mode, α is the intercept, which refers to the value y when x is zero; β denotes the regression coefficient or slope, which indicates the rate of variation in y for one unit variation in x, and i is the observation record, and ϵ is the error term. In fact, in practical research, we have an error of estimate for y. The residual

$$\epsilon_i = y_i - \overline{y_i}$$

is the difference between the value of the dependent variable predicted by the model, $\overline{y_i}$, and the true value of the dependent variable, y_i. Given the population sample of independent variable x and y, where $\bar{x}$ denotes the mean of sample population of x, and $\bar{y}$ denotes the sample population of y, then the regression coefficient is obtained as [71]:

$$\beta = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

and mean α is obtained as:

$$\alpha = \bar{y} - \beta\bar{x}$$

As mentioned above, multiple regression is done for several independent variables (IV) as predictors (i.e. Activities clustered from event logs), and one dependent variable (DV) (i.e. a monitoring metric) as the outcome. An objective of applying regression technique is to derive a model from input sample data with the minimum absolute (squared) error [99]. Given:

- y is the dependent variable,
- $\beta_1, \beta_2, \ldots, \beta_n$ are the regression parameters,
- $x_1, x_2, \ldots, x_n$ are the independent variables,
- ϵ is the error term or noise,
- α denotes a constant value as an intercept, where it indicates the mean value of y when all $x=0$.

The general form of a multiple linear regression function is

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

The coefficients $\beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$ 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 for which they are suitable. A statistical model often can be generated by using standard statistical software packages such as R,[15] SPSS,[16] SAS,[17] STATA,[18] Minitab[19] and so on. We used mainly SPSS to conduct our experimental data analysis.

Multiple regression is a robust model, used as the basis of data analysis in many disciplines [99]. In the next sections, we will employ this technique for finding the correlation between resource metrics and activity logs, and, as a result, select most relevant metrics to the changes of log events. Then, we further leverage the exploratory aspect of multiple regression for identifying the most influential factors that change the resources. Finally, we take the outcome of our exploratory analysis to derive assertions and use assertion formulas for online anomaly detection.

3.4.1 Target Metric Selection

In system monitoring, there are usually many monitoring metrics available. For example, AWS CloudWatch offers 168 metrics for an ASG with 10 machines, an ELB and Elastic Block Storage (EBS).[20] When administrating dozens 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 [94, 109, 138]. Other problems include a system operator receiving too many false alarms or a flood of alerts

[15] http://www.r-project.org/
[16] https://www.ibm.com/analytics/au/en/technology/spss/
[17] http://www.sas.com/
[18] http://www.stata.com/
[19] http://www.minitab.com
[20] At the time of writing this thesis, 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

from different channels about the same event [12].

In today's practice of system monitoring, the task of metric selection is done by system operators, mostly manually, based on their domain knowledge. In fact, most of the software monitoring packages, such as Nagios, Sensu[21] and Misto[22] provide customisation, often allowing the choice of which metrics to show and how alarms should be generated. However, this customisation has become more difficult with the ever-growing number of resource types in cloud environments and the variety of associated metrics.

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 a nutshell, reducing monitoring metrics dimensions has the following advantages:

- it helps system operators to focus on a limited number of metrics, and thus, they can have a faster response to detected anomalies [57].
- it reduces the exposure to too many alarms, and, as a result, reduces alarm fatigue [94, 109, 138].
- it reduces the high cost of system monitoring by narrowing down the focus of monitoring to the most relevant metrics, in other words, smaller number of metrics to monitor reduces the volume of monitoring data [2, 20, 38, 62, 67, 91, 120].

We developed a new approach that facilitates the process of metric selection, primarily by statistical analysis, rather than merely relying on domain knowledge. 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 predictive abilities.

In our approach, we refer to metric selection as a process of identifying a subset of metrics that can be helpful to get the optimum anomaly detection result while considering both logs and metrics. Thus we aim to identify, from all

[21] https://sensuapp.org
[22] https://mist.io/

original available metrics, a subset of metrics that have the highest correlation with the activities of logs. We are also interested to present these metrics with the highest to the lowest relevance. We can perform the above task with the help of our multiple regression analysis.

In the previous section, we explained that a regression model helps us to model the correlation relationship between a set of independent variables and a dependent variable. In the process of metric selection, independent variables are our log activities, and the dependent variable is the candidate metric. The prerequisite step to learn the impact of the behaviour of event logs on resources and select target metrics is to observe the behaviour of a system over a period of time and collect logs and metrics within this period and map these two sources of data, as explained in previous sections.

To achieve robust learning from statistical analysis, it is important to collect a large enough number of data records. As a rule of thumb, having n number of independent variable (log activities), $(n * 10)$ records or data needed [121]. For instance, assuming having 20 activities as the result of log abstraction, then $20 * 10 = 200$ records of data will be a desirable number. One may still apply regression analysis with a smaller number of records and find sensible results, however, the higher the number of observation records of data, the better the accuracy of the regression model [71] will be.

Unlike many other fields, such as in the medical or social domain, where collecting sample data is often a timely and costly process, in the domain of system health and performance monitoring, historical monitoring data are often collected continuously and are available in a large volume [60]. Thus, many data-oriented approaches such as ones that employ statistical and machine-learning methods, have shown to be effective in producing robust models in this domain [60].

Given a sufficiently large number n of records of data of a sample population, independent variables as predictors, denoted by $x_1 \dots x_p$ (log activities), the *observed* dependent variable, denoted by y_i (actual value of metric), the *estimation* of y_i using the regression model (also called the *fitted response*), denoted by f_i (predicted value of the metric), with $\bar{y}$ being the mean of the

observed dependent variable and $\overline{f_i}$ the mean of the estimations is:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{3.1}$$

The total sum of squares is obtained as:

$$SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{3.2}$$

Similarly, the sum of squares of residuals is defined as:

$$SS_{res} = \sum_{i=1}^{n}(y_i - f_i)^2 = \sum_{i=1}^{n}\epsilon_i^2 \tag{3.3}$$

Using equations 3.2 and 3.3, we can determine how much variation of a dependent variable can be explained by a predictor. The coefficient of determination R^2 is defined as [99]:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{3.4}$$

R^2 indicates how well a model predicts new observations, and can be used to assess the predictive power of a regression model for the given predictors and target variables [99], respectively. $Adj.R^2$ is a slightly more conservative version of R^2 that penalizes a high number of predictor variables in a model [99]. $Adj.R^2$ is always equal or less than R^2 and the difference between R^2 and $Adj.R^2$ gets smaller as the sample size increases. With p being the total number of independent variables in the model and n is the sample size, $Adj.R^2$ is defined as:

$$Adj.R^2 = 1 - \frac{n-1}{n-p}(1 - R^2) \tag{3.5}$$

It is important to note that a high value for R^2 indicates that a metric has a linear relationship with the activity logs of an operation, and such a metric can be *potentially* employed for 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 regression

models. In two case studies presented in Chapters 5 and 6, we will show how we applied this technique to narrow down the number of metrics to the most sensitive metrics of the activities of event logs.

3.4.2 Identification of Influential Log-Events and Assertion Derivation for Anomaly Detection

The core objective of this thesis is to present an approach to improve system dependability through system monitoring and anomaly detection. We have the objective to present an anomaly detection mechanism with the following characteristics:

- to be a non-intrusive method – meaning it would not require modification of existing monitoring systems, and it should be utilised on top of the existing systems;
- to be an unsupervised method – meaning it would not require anomalous data labels for learning data;
- to be a context-aware method – meaning it would include the aspect of behavioural utilisation of the system on anomaly detection;
- to be a real-time/online method – meaning the proposed technique should be used with live data, rather than through a post-mortem analysis;
- to be a method with low computational complexity – monitoring data is continuous and voluminous, therefore we intended to avoid methods with high computational complexity at run-time.

In this direction, so far we have presented logs as a quantitative form, using the derived interpolated occurrence strength of log events (3.3.1), abstracted the log events to a set of activities (3.3.2), mapped activity logs to resource metrics, and identified the most relevant monitoring metrics without breaching the above features (3.4.1). As the next step, we focus on employing regression-based analysis to identify from event logs the influential activities that affect resources. One of the objectives of performing multiple regression analysis is

to find an explanatory relationship between independent variables (activities) and the dependent variables (resource metrics).

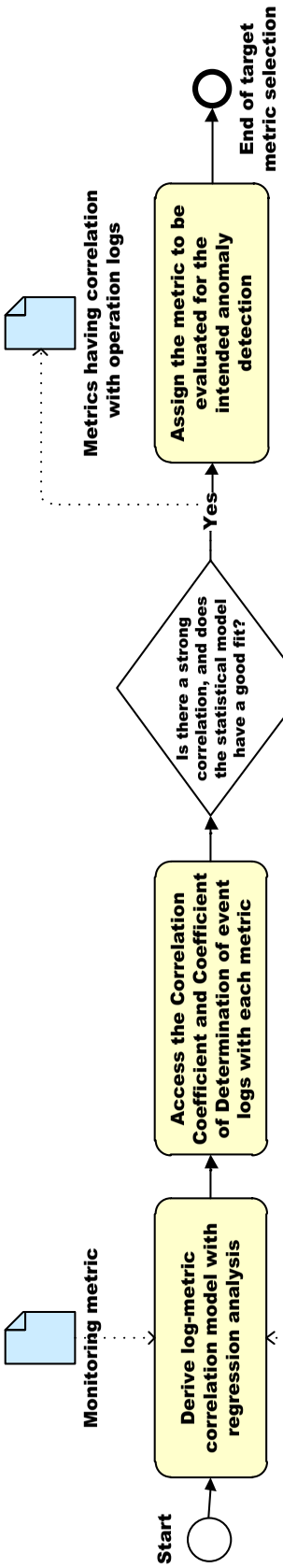


Figure 3.6: Checking the relevancy of a monitoring metric

"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 [59]. It is important to note that while the presence of causation certainly implies correlation, the existence of correlation only implies a potential causation. For instance, one may observe a correlation between power consumption and 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 are involved in a scaling-up process to respond to 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.

To give an example, one of the best practices of upgrading applications in environments with virtualisation technologies is to prepare a new VM image that contains the new version of the application. Then, the new image, in an upgrade process, is deployed to the cloud by terminating the old VM instances and replacing them by launching new VM instances that had been created from a new image. Hence, 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," followed by the log events that report the launching process of a new VM instance. As the example suggests, there will be a linear correlation between log events of the launching VMs and the termination VMs; however, does that imply log events of launching instances caused the termination? Of course not, and here we are interested in leveraging the powerful explanatory feature of regression analysis to distinguish these two.

In the previous section, we explained how to find the metrics that have correlations with the operation's activities overall, yet we need to find out which of the *operation's activities* are affecting a target metric to derive assertion

specifications for an anomaly detection requirement. We thus seek to distinguish the activities of the operation that are likely to affect one of the target metrics from the others.

In order to perform such analyses, we consider the coefficient correlation result from the regression analysis, where y denotes the dependent variable (metrics), and x_1, x_2 and x_3 refer to the relevant activities from logs, also called predictors. A model of coefficient output from OLS regression analysis is shown in Table 3.2, and Table 3.3 explains the list of parameters of coefficient correlation output.

The resulting regression equation derived from Table 3.2 is:

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

From the model, we learn concrete values for α and the β_i. 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 $\beta_i = 0$. The $p - value$ gives us a measurement criterion to make sure a predictor is being *statistically significant* in our statistical observation. Moreover, the standardised coefficient will tell us how much a predictor contributes to the changes of a metric.

Table 3.2: Coefficient correlation output of OLS regression

Predictors	β	Std. Error	B	*p-value*
Intercept (Constant)	β_I	se_I	—	p_I
x_1	β_1	se_1	B_1	p_1
x_2	β_2	se_2	B_2	p_2
x_3	β_3	se_3	B_3	p_3
$x_{..}$	$\beta_{..}$	$se_{..}$	$B_{..}$	$p_{..}$
$x_{..}$	$\beta_{..}$	$se_{..}$	$B_{..}$	$p_{..}$
x_n	β_n	se_n	B_n	p_n

If the standardised coefficient value is close to zero for a predictor, it means

Table 3.3: Coefficient Correlation Table Notations

Notation	Description
α	Intercept - also labelled as constant
Predictors	Activity log events
β	Unstandaridized regression coefficient
Std. Error	Standard error of estimate
B	Standardized regression coefficient
p value	Calculated probability, where $p > .005$ is statistically significant

that the predictor has almost no impact on changes in a resource. Hence its effect on the assertion specification equation will be none. For instance, activities that report waiting or a status rather than reporting an action, have no effect on a resource. Other activities recorded in logs might have an impact on one metric but not on another. For example, an activity may have a high impact on Disk I/O but an insignificant impact on network metrics. Therefore, by leveraging regression analysis and assessing the standardised coefficient values, the number of predictors can be narrowed to the only ones that actually contribute to the changes in resources.

In order to do anomaly detection 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) is obtained. Every minute (or other time-interval), a prediction is calculated and compared with the actual value of the metrics. The above outcome can be used for live anomaly detection of application operations.

One challenge remains: as the prediction is not usually as exact as the actual value, a threshold of accepted range should be considered. To define a threshold t, such that $0 < t < 0.5$, 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).

3.5 Closely Related Work to the Anomaly Detection Approach of this Thesis

Previously we presented and discussed the concept of anomaly detection, explained the anomaly detection characterisations classifications. Especially, we gave an overview of common anomaly detection techniques and highlighted strength and weakness of each techniques. In this section, we focus on the studies that are closely related to the approach of this thesis.

3.5.1 Context-based Anomaly Detection

Anomaly detection has been broadly employed for system health and performance monitoring, nevertheless, the majority of the anomaly detection approaches have mostly focused on point-based techniques [60]. In this type of technique, the anomaly detection is conducted based on the monitoring of a target metric and comparison of the metric with the rest of data. However, point-based techniques do not take into account the impact of dynamic nature of workload or the legitimate contextual and behavioural factors are causing anomalous spikes on system resource utilisation. This issue motivated several studies to propose context-based techniques. Anomaly detection in context-based techniques is conducted according to a set of conditions or behavioural attributes of a system.

In fact, a point anomaly detection problem can be transformed to a contextual anomaly detection problem by incorporating the context information [23]. The principal advantage of contextual anomaly detection approaches is that they are based on a natural model of an anomaly in many real-life applications where data instances tend to be similar within a context [19]. These approaches can find the anomalous cases that might not be identified by point-based anomaly detection techniques due to taking a global view of the data for anomaly detection [19].

The most utilised methods of contextual anomaly technique in the domain of system health and performance monitoring is using time-related factors or other simple conditions as a set of rules for assessing the normality of a data

point. For instance, a high volume of system transactions towards Christmas is considered normal, while it is not normal at other times of a year.

While contextual anomaly detection techniques have been most commonly studied in time-series data [23], there have been a small number of studies making use of other sources of information. The context-based anomaly detection methods have been highlighted as one of the main future directions in anomaly detection [60], especially, this gap has become more significant by the nature of cloud computing environments, where a higher level of flexibility and dynamism is provided.

In this direction, this thesis attempts to address this gap by proposing a context-based anomaly detection through a novel combination of both resource metrics and event logs. Therefore, in the following we focus on reviewing the past studies that attempted to contribute to this domain by employing information that reflect the dynamic behaviour of systems in combination with resource metrics. In addition to discuss each of past studies, we provided a comparison view of the above-discussed related work to this thesis at the end of this section, presented in Table 3.4.

3.5.2 Closely Related Work

Few studies model system behaviour through workload profiling and then combine this information with resource metrics for anomaly detection. Wang et al., propose an approach using an incremental k-means clustering technique to recognise access patterns and request volume from the workload [127]. Then local outlier factor (LOF), has been employed to identify anomalous data instances for each type of workload pattern. LOF is a machine learning technique which works based on a concept of a local density. In this method, locality is given by k nearest neighbours, where distance is employed to estimate the density. By measuring the local density of a data point in comparison to the local densities of its neighbours, the areas of similar density can be detected, and objects that have a considerably smaller density than their neighbours are tagged as anomalies.

The above approach has the advantage of being independent of domain

knowledge. However, their approach has the limitation on workload characterization as their method is not suitable for fine-grain monitoring due to many clusters generated from workload patterns [128]. Also, this approach has high computational complexity due to workload pattern recognition and LOF calculation for each arrival data instance. And in contrast to our work, their approach was a supervised approach, and hence required labelled data.

In an extension to above work, Wang et al., [128] attempted to address the above limitations. Wang et al., [128] proposed an approach where the signature of different workload on resource utilisation is analysed using Canonical Correlation Analysis(CCA) technique. Then, by identifying the abrupt changes of correlation coefficients with a control chart, anomalous data points are detected, and then this information along with a feature selection method is employed to suggest suspicious metrics that are associated with the detected failure.

Similar to our work, this approach can be utilised for unsupervised data, which does not need anomalous data instances to be labelled. Also, their approach could handle fine-grain workload characterization. Nevertheless, workload profiling used in Wang et al., study is limited to the number of concurrent users, response time, and throughput. In contrast to our work, the type of context of workload has not been taken into account, therefore, their approach is incapable of including the influence of the workload at activity-level.

In another study, Cherkasova et al., [25] present a regression-based model to model the resource consumption of Web applications. They present a profiling method by identifying application performance signature (using transaction count, transaction latency, count of the database call, and latency of outbound call) in order to model the run-time application behaviours. Their work is shown to be effective in detecting CPU consumption pattern; however, no other metrics have been explored.

In comparison to our work, similarly, they employed a regression-based technique, but our work differs from the above from the following viewpoints. One limitation to their approach is related to the adopted application performance modelling method, as their method for application performance model

is solely based on transaction volume and latency, while in our approach we focus on event logs as the indicators of workload. Also, it is not clear whether the above approach can be applied to state-based metrics, besides, CPU was the only non-state based metric that has been examined in their study. In addition, the exploratory/causation aspect of regression model has not been used, as the way we do for identifying the log activities that stress a resource the most and for the derivation of assertion specification.

Magalhaes and Silva, [80, 81]; introduce an approach to detect root-cause factors of observed performance variations due to workload changes or application updates. This is done by adopting the Pearson coefficient of correlation between system metrics and aggregated workload. In this approach, they employed Aspect Oriented Programming (AOP) to monitor the response time of every transaction and then Pearson correlation is used to model the correlation between transaction and response time.

Magalhaes and Silva's approach is limited to work with just one metric of workload ([69]) at a time. This is a considerable weakness with the common large scale web applications that perform hundreds of transactions per second, or more. Moreover, monitoring response time with changing source code enforces significant overhead to the monitoring system. Lastly, it is an intrusive approach as it requires access to the native code of the application.

Kang et al., [65] proposed DAPA (Diagnosing Application Performance Anomalies), a statistical approach to model the quantitative relationship between the application response time and virtualized system metrics based on SLA violations. The main criterion in their study for monitoring anomalies were the indicators of SLA violation. In the above approach, anomaly detection target metrics are not system resource metrics, application response time is the only source of monitoring information for detecting SLA violation. Being limited to one metric of application response time, this approach has the limitation to detect failures that do not lead to immediate performance degradation.

As one of the recent studies close to the idea of this thesis, Gurumdimma et al., [53], propose CRUDE (Combining Resource Usage Data and Error Logs),

to detect errors. Similar to our approach, they take advantage of both application logs and resource metrics. CRUDE methodology relies on the computation of mutual information, entropy and anomaly score to identify the chain of events that may lead to a failure. Logs are analysed with hierarchical clustering and feature extraction methods, and Principal Component Analysis (PCA) is employed to detect anomalous jobs within a period. CRUDE assume that a higher entropy (uncertainty) with reduced mutual information could denote abnormal system behaviour or a failure sequence, with the opposite signify the normal behaviour.

Although this approach is similar to our study with regards to using both logs and metrics for anomaly detection, it is very different in terms of the proposed framework. Firstly, the approach detects sequence anomaly (rather than point-anomaly) based on the sequence of events in a relatively long time-window (e.g. 60 minutes), while in our study we aim to detect point anomalies in the relatively small time window (e.g. 5 seconds or 1 minute). In other words, this approach identifies the faulty sequence of events preceding failures based on changes in the entropy of sequences, instead of detecting individual fault events. Secondly, setting thresholds for various metrics needs multiple observation and readjustment to find an optimum result, while in our approach the threshold is set (though configurable) based on the error estimate of the regression model. Their approach are shown to be useful in HPC data centres, but it is unlikely to deliver a lot of value in cloud settings. This limitation makes defining thresholds for a large number of metrics a cumbersome task. Besides, the effectiveness of the approach is not examined with resource metrics, alternatively the resource consumption for each computing job is leveraged as the metrics under observation, such metrics profile may not be available in many systems.

One of the studies that inspired the work of this thesis, which we address its limitations to some extend, is called *POD-Diagnosis* [23] [140]. This ap-

[23]POD-Diagnosis and POD-Monitor are part of the work of Process Oriented Dependability(POD) research group in Data 61 (https://research.csiro.au/data61/process-oriented-dependability/), which I am a member of it and thus I had the chance to receive their guidance and disscuss their work. Also I had access to their data and resources, and the anomaly detection tool that I developed has the architectures in a way to be integrated with other

proach 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 [137, 140]. 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 [140], 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 assertions 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 typically 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 POD-Diagnosis as we rely on statistical correlation analysis rather than domain knowledge.

The second limitation is related to the dependency on logs as the main source of information for operation monitoring.

POD-related tools of this team.

- First, logs are often low-level, noisy, and with inconsistencies in style [95]. For instance, in [95] the authors report the difficulties of failure detection due to a lack of relevant information in logs, and [29] 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 [143].

- 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 makes the mapping between these two more challenging.

- third, monitoring execution behaviour 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 other running operations.

These limitations exacerbate the difficulty of error detection for cloud operations, and relying on log content limits the generalizability 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 leverage cloud metric data, in addition to information extracted from logs, to cross-validate the execution of cloud DevOps operations.

Another approach that attempts to address the above limitations is POD-Monitor [138]. Xu et al., attempt to address the gap of monitoring DevOps operations in the cloud environment by using a process model extracted from logs and data point metrics. In their technique, a process model that is the indicator of start time, progression, and stop time of an operation is used as a contextual information to suppress false alarms of detected anomalies

in resources usage. However, their approach did not track individual event logs and their approach lacks the support for anomaly detection of steps of cloud operations that are proposed in this thesis. 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.

The domain of anomaly detection is vast, and some studies have attempted to conduct systematic surveys in this domain [3, 23, 56, 60]. One of the highlighted issues in the domain of system monitoring is the lack of cross-layer monitoring [2]. Cross-layer monitoring is a challenging task, as it is difficult to map different monitoring data types and to interpret them in an integrated form. This thesis, in particular, contributes in this direction, as we consider two different types of monitoring information, which can span multiple layers. Thereby, we narrowed down our focus to the context-based anomaly detection methods in system health and performance monitoring, which attempted to combine different sources of information for anomaly detection.

In order to give a comparison view of the above-discussed related work to this thesis, we summarised our analysis in Table 3.4.

Paper	**Monitoring Resource**	**Method**	**Technique**	**Intrusive /Non-Intrusive**	**Virtualisation /Cloud**	**Application Domain**	**Supervised /Non-Supervised**
Cherkasova et al., 2008 [25]	System-level metric(only CPU), Application Workload	Statistical	Linear Regression (Predictive Analysis)	non-intrusive	No	Web-based Multi-tier application	Unsupervised
Gurumdimma et al., 2016 [53]	Job-resource metrics,lLogs	Information Theoretic, Statistical	Clustering Logs using (distance from centroid), Information Theoreic(entropy), Statistical(PCA)	intrusive	No	Hadoop	Unsupervised
Kang et al., 2012 [65]	system-level metrics, aplicaiton response time	Clustering and Statistical	K-means Clustering (for SLA model clustering) Regression (for metric and SLA relationship)	non-intrusive	Yes	Non-Web Application on VMs	Supervised

Paper	Monitoring Resource	Method	Technique	Intrusive /Non-Intrusive	Virtualisation /Cloud	Application Domain	Supervised /Non-Supervised
Magalhaes and Silva, 2010,2012 [80, 81]	system-level metrics, Workload from Transcation Mix model.	Statistical	Correlation Analysis, ANOVA	intrusive (request/response tracing with native code)	No	Web Application	Unsupervised
Wang et al., 2014 [127]	System-level metrics, Web Application Workload	Clustering	Clustering (Incremental Clustering Method and K-Means For workload pattern recognition - Unsupervised) and Nearest Neighbor(LOF for anomaly detection -Supervised)	non-intrusive	No	Web Application	Supervised

Paper	**Monitoring Resource**	**Method**	**Technique**	**Intrusive /Non-Intrusive**	**Virtualisation /Cloud**	**Application Domain**	**Supervised /Non-Supervised**
Wang et al., 2016 [128]	System-level metrics, Web Application Workload	Clustering and Statistical	Clustering (Incremental Clustering Method and K-Means For workload pattern recognition - Unsupervised) and Statistical (CCA for anomaly detection -Unsupervised)	non-intrusive	Yes	Web-Application in Cloud	Unsupervised
Xu, et al., 2015 [138]	System-level metrics, logs	Classifiction	Machine Learning - Support Vector Machine (SVM)	non-intrusive	Yes	Cloud, DevOps Operation	Supervised
This thesis approach	State-based and Non-state based resource metrics, logs	Statistical	Pearson Correlation Clustering for Logs, Regression (Predictive and Exploratory Analysis) for Anomaly Detection	non-intrusive	Yes	Non-Cloud and Cloud, DevOps Operation	Unsupervised

Table 3.4: Studies adopted multiple source of information for anomaly detection in system health and performance monitoring

3.6 Summary

In this chapter, we described a conceptual framework of a metric selection and anomaly detection technique, along with the steps of the proposed approach. Initially, we gave an overview of the approach, where the relationship between log events and resource metrics for system monitoring was described, and a workflow of the proposed approach to anomaly detection, using both logs and metrics, was presented. Then, we explained the steps of the workflow, including the techniques that this study has utilised to extract metrics from logs, and to cluster logs to activities using Pearson's correlation coefficient. Also, this chapter has described the mechanism to identify the statistically-proven relevant target monitoring metrics from all available metrics. Finally, we discussed our proposed approach of deriving assertion specification and identification of influential factors from log events.

The proposed approach of mapping log events to resource metrics, selection of target metrics, identification of influential factors in the logs, and assertion equation extraction from regression analysis, provides a framework that facilitates integrated cross-level (logs and metrics levels) monitoring and anomaly detection. This is achieved by comparing the behaviour of system operations that is reflected through the operations' logs with the status of the resources that are reflected in the metrics.

Further in this chapter, we reviewed and discussed the contextual anomaly detection studies that attempted to incorporate a source of information other than resource metrics data for their anomaly detection. In this direction, a brief summary of each method and its limitations along with the differences of each approach in comparison to our approach is presented. Also, we presented a review of past studies on metric selection and discussed the strength and weakness of each approach in relation to the metric selection method proposed in this thesis.

In the next chapter we will introduce the tool that we developed to conduct the experiments of the proposed approach, followed by two industry-grade case study chapters that we will perform the experiments and evaluate the effectiveness of the approach presented in this chapter.

Chapter 4

Anomaly Checking Prototype

The anomaly detection approach of this thesis can be divided into two main phases: learning and testing/evaluation. In the learning phase, an anomaly detection model from training data is built, according to the information contained in the logs and metrics data of the project under study. Once the model is built, it can be used many times for anomaly checking.

In this direction, we focussed our efforts on developing a prototype that automates the anomaly checking process at run-time, as well as for post-mortem analysis. The tools and techniques that are used during the learning phase are explained with examples of actual data of two separate case studies in Chapter 6 and 7, respectively. This chapter concentrates on the prototype that is used for testing/evaluation phase.

In this chapter, we introduce this prototype, together with its features and architecture. On the basis of this prototype, the ideas proposed in this thesis are evaluated with two industry-grade case studies that are presented in Chapters 6 and 7, respectively.

4.1 Anomaly-Checker and Integration with POD Services

Anomaly-Checker is implemented as an independent micro service of the bigger architecture of Data61 Process Oriented Dependability (POD) services [1].

[1] https://research.csiro.au/data61/process-oriented-dependability/

POD refers to a set of activities, tools, and services that aim to improve the dependability of process operations, with a focus on cloud application operations. "POD treats operations as processes and uses a process context to provide a basis for run-time error detection, diagnosis, and recovery" [32].

As part of the POD research effort, several solutions have been proposed, including POD-Discovery [129, 130] that addresses discovering processes from logs/scripts, POD-Detection that focuses on error detection among others by process conformance checking [131, 140], POD-Diagnosis that is designed to automatically diagnose errors using fault trees, Bayesian networks and automatic diagnostic testing [139, 140], POD-Recovery which aims to perform guided/automatic recovery [42], and POD-Viz [130] that handles the visualization of operation progress and error detection. As a result of the above activities, a few services and tools have been developed, which have been partly released [32].

Although our Anomaly-Checker, as introduced in this chapter, is designed to work as a standalone service, its architecture is integrated with some of the other POD services. The anomaly detection output of Anomaly-Checker can be used for data visualisation through the user interface provided by POD-Viz. Anomaly-Checker provides the results in JSON format and the POD-Viz processes this input for appropriate visualization of logs and metrics. Further, when anomalies are detected by Anomaly-Checker, it has the ability to automatically trigger the services of the POD-Diagnosis tool for error debugging and diagnosis.

Further information about POD research activities, publications and tools can be found on the link provided in the Footnote 1 in the previous page.

4.2 Key Features and Architecture

Anomaly-Checker is a service-oriented application that is developed for detecting anomalies in a system, based on the data from activities on logs and resource monitoring metrics, respectively. To this end, Anomaly-Checker provides the following key features:

- data access management for monitoring data (store, retrieve/search, update, and delete)

- tracking and identifying log event types, counting the occurrence of log events within a time window and deriving the metric value for each log event

- performing run-time anomaly detection based on assertion checking between logs and metrics, and reporting the status of anomaly checking

- appropriate web service interfaces to provide Anomaly-Checker services to POD-Viz and other application services.

To provide the above features, we designed and developed Anomaly-Checker using the Java and MVC (Model View Controller) architectural design pattern. MVC is a widely adopted design pattern that provides separation of concerns by differentiating the layers of a project. MVC facilitates future enhancements, re-usability of the code, and maintainability of applications. Although our main objective was to develop a simple prototype, we chose the Spring MVC architecture pattern because of the above features, in order to allow the prototype to be easily enhanced and integrated with other POD services. Also, we used RESTful web services, as they make the functions of the prototype available as web services that can be easily called by POD-services or other third-party applications.

The abstract architecture of Anomaly-Checker is shown in Fig.4.1. Anomaly-Checker is a service-oriented application that has a multi-tier architecture comprising the following abstract layers:

- Persistence layer: This layer stores metric and log data in a persistent storage and also stores the results of anomaly checking. We used the Elasticsearch database [2] as the data repository in our project.

- Data access layer: This layer provides a mapping application for the persistence layer and simplifies access to the stored data. We adopted

[2]https://www.elastic.co/products/elasticsearch

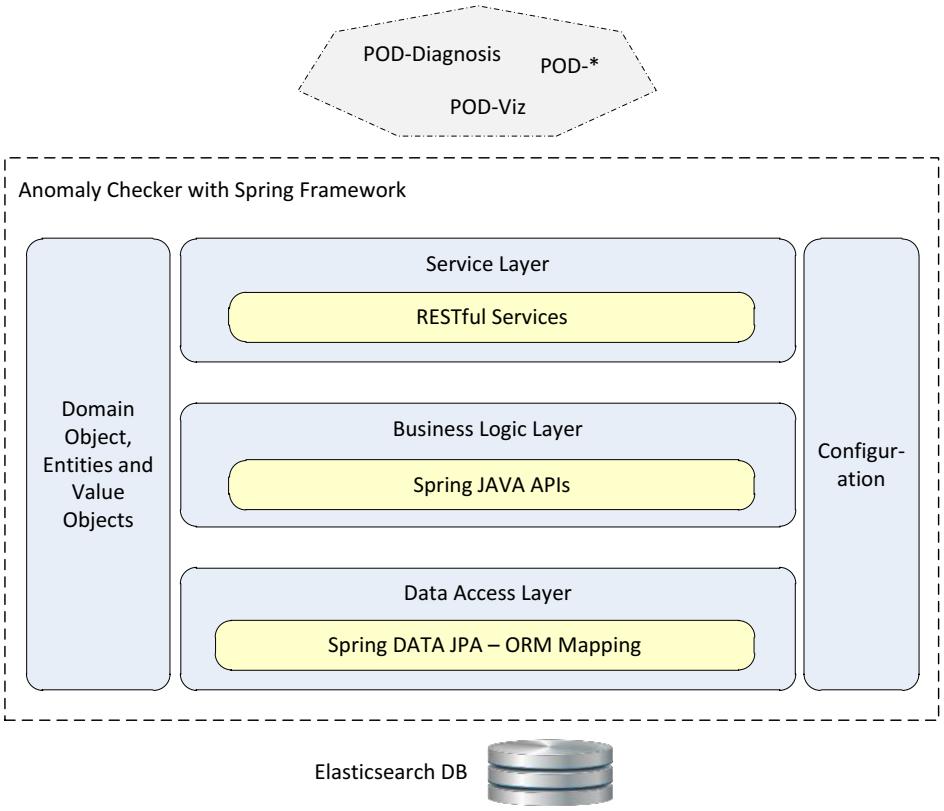


Figure 4.1: Anomaly-Checker Abstract Architecture

Spring Data JPA[3]to implement this layer. Spring Data provides rich APIs to access data repositories and expose data to upper layers.

- Business logic layer: This layer encodes the core logic of Anomaly-Checker. It defines the business rules of how to create, store, and access data, as well as the logic, to perform the anomaly checking itself.

- Service layer: This layer is responsible for exposing the Anomaly-Checker application functions to external application services. All the implemented services are developed with RESTful technology. REST has the advantages of being independent of platform and language, provides decoupling between the client and server, and can be directly called through HTTP URL fetching. This feature allows Anomaly-Checker services to be easily integrated with other services.

[3]https://projects.spring.io/spring-data-jpa/

4.2.1 Data Repository: Elasticsearch

Anomaly-Checker requires two sources of data inputs: monitoring metrics and log events. As different monitoring solutions provide access to monitoring data with different technical specifications, we chose to have an independent data repository for the monitoring data from logs and metrics. We used a database called Elasticsearch as our monitoring data repository for the reasons outlined below.

Elasticsearch is an open source non-relational data storage that stores our monitoring and anomaly detection data. We preferred Elasticsearch as a non-relational database rather than a relational database, as Elasticsearch natively stores data in JSON format, and because this format is being commonly used in the monitoring solutions of public cloud providers such as Amazon CloudWatch and Google Stackdrive. Having a format similar to mainstream monitoring solution providers in the cloud facilitates the task of data conversion and integration between Anomaly-Checker and these monitoring services. Also, Elasticsearch is not just a database that stores the monitoring data, but it is also a search engine that provides a distributed, multitenant-capable [13] full-text search engine. Elasticsearch is well integrated with Kabana[4] (an open source data analytics and visualisation solution), and Logstash[5] (an open source data processing pipeline that ingests data from multiple sources). In addition, the data stored in an Elasticsearch database can be directly viewed and updated by an HTTP-based web interface using schema-free JSON documents. These features makes Elasticsearch an ideal tool to be used as the data repository in our Anomaly-Checker prototype, as well as to be used for extra data analysis of monitoring data when needed.

4.2.2 Data Schema

We designed three data schemas for Anomaly-Checker in our Elasticsearch database: a schema to keep the resource monitoring data, a schema to store the operation's log events, and a data schema that keeps track of anomaly

[4]https://www.elastic.co/products/kibana
[5]https://www.elastic.co/products/logstash

Metric Schema


```
{
   "properties":{
      "id":{
         "type":"string"
      },
      "dimensions":{
         "type":"string",
         "index":"not_analyzed"
      },
      "label":{
         "type":"string"
      },
      "experiment":{
         "type":"string"
      },
      "timestamp":{
         "type":"long"
      },
      "timestampFormatted":{
         "type":"date",
         "format":"dateOptionalTime"
      },
      "average":{
         "type":"double",
         "index":"no"
      },
      "maximum":{
         "type":"double",
         "index":"no"
      },
      "minimum":{
         "type":"double",
         "index":"no"
      },
      "sampleCount":{
         "type":"double",
         "index":"no"
      },
      "sum":{
         "type":"double",
         "index":"no"
      },
      "unit":{
         "type":"string",
         "index":"no"
      }
   }
}
```


Anomaly Report Schema


```
{
   "properties":{
      "actualValue":{
         "type":"double"
      },
      "anomalyStatus":{
         "type":"boolean"
      },
      "anomalyStatusConfidence":{
         "type":"string"
      },
      "dimension":{
         "type":"string",
         "index":"not_analyzed"
      },
      "estimatedValue":{
         "type":"double"
      },
      "estimatedValueDiscreted":{
         "type":"double"
      },
      "id":{
         "type":"string"
      },
      "label":{
         "type":"string"
      },
      "logOrigin":{
         "type":"string"
      },
      "logs":{
         "type":"string",
         "index":"no"
      },
      "timestamp":{
         "type":"long"
      },
      "timestmapFormatted":{
         "type":"date",
         "format":"dateOptionalTime"
      }
   }
}
```


Log Event Schema


```
{
   "properties":{
      "log":{
         "type":"string",
         "index":"analyzed"
      },
      "logTime":{
         "type":"long"
      },
      "logSeqNum":{
         "type":"long"
      },
      "logOrigin":{
         "type":"string",
         "index":"not_analyzed"
      },
      "processModelId":{
         "type":"string",
         "index":"not_analyzed"
      },
      "processInstanceIds":{
         "type":"string",
         "index":"not_analyzed"
      },
      "activityId":{
         "type":"string",
         "index":"not_analyzed"
      }
   }
}
```


Figure 4.2: Elasticsearch Data Schema for Metric, Anomaly Report, and Log Event

checking reports. The schema definitions are shown in Fig. 4.2.

The metric schema is designed to store the raw monitoring data for each metric. Each metric is presented in the schema by its *label*. Each metric belongs to a *dimension* that refers to a node or a name-value pair that uniquely identifies a metric such as a VM instance or a load balancer. In cases where the metric is a cumulative metric (by having multiple sample observations since the last timestamp), then we may have a minimum, average, and maximum, sample count, and a sum as the values for the metric, similar to the Amazon Cloudwatch data format. The average value is taken as the default value of

the metric. In this schema, "unit" specifies the unit of the sample value, such as percentage for CPU or byte for disk I/O.

The log event schema is designed to keep the records of log messages. Before log messages being stored to this schema, they need to pass a pre-processing step which makes sure each log message contains a timestamp, and thereby it extracts the timestamp from the log. These data are stored in *log*, and *logtime*, respectively. *logOrigin* refers to the identifier of an operation that the log originated from and *activityId* is the indicator of which activity this event is clustered to. The other fields in this schema are reserved for consistency with other POD-services.

The anomaly report schema brings the log and metric into an integrated view. Looking at the anomaly report schema in Fig. 4.3, it contains the information of the estimated value (*estimatedValue*) from log events, the actual value (*actualValue*) from the resource metric, and the result of the anomaly (*anomalyStatus*). In addition, we provided the metric and logs involved in the anomaly checking report. Two samples of the anomaly report are shown in Fig. 4.3.

4.3 Configuration Input

Anomaly-Checker, in addition to logs and metrics as input, requires a set of configurations to perform anomaly checking. The following are the configuration inputs:

- General Spring application configurations using native format of the framework in ".properties" files to boot and execute the application. The configuration includes the URL and port settings, Elasticsearch access DB settings and other related application settings.

- Configurations related to log processing with regular expressions. We use a simple text file to define the regular expressions listed in separated lines. Each regular expression is unique and represents the pattern of a unique log event type. These regular expressions are derived for each log origin

through log processing, as described in Section 3.3 (also demonstrated with examples for two case studies in Sections 5.2 and 6.2).

In our Anomaly-Checker, each log message is compared against the list of regular expressions to find the matching log event type. Based on this matching process, we track the occurrence of logs and derive the interpolated occurrence strength of each log activity, which is used, along with coefficient values, to build the assertion formula.

- Coefficient values derived from an assertion equation are the other required configuration inputs. The assertion formula is obtained from regression analysis during the learning phase, as described in Section 3.4.2 (also demonstrated with actual values from investigating two case studies in Sections 5.4 and 6.4). To estimate the value of metrics from log occurrences at run-time, we need to know the coefficient value of each log activity to obtain the estimated value of a metric.

  We designed a matrix presented as a CSV configuration file for this purpose. In our matrix, each row indicates the identifier of log activity and each column identifies the target metric. The matrix cell contains the coefficient value related to the log activity and metric. The assertion equation is automatically built from the given matrix in the implemented code and used by the anomaly checking service at run-time.

4.4 Anomaly Checking

In the Anomaly-Checker prototype, all the main use cases of the prototype are exposed as web services. In addition, the Anomaly-Checker services are designed in a way that can be simply called by issuing an HTTP request. The data of calling the Anomaly-Checker services are returned in JSON format.

In our prototype, ultimately we aimed to perform anomaly checking, but first, we had to implement the necessary operations to manage access to monitoring data in our data repository. Therefore, several operations were implemented to perform CRUD actions, as well as a few searching operations to

```
{
   "_index":"pod",
   "_type":"anomalyReport",
   "_id":"1415228280000_Asgrad-Rolling-Upgrade-Log_AutoScalingGroupName:testworkload-r01_CPUUtilization",
   "_score":1,
   "_source":{
      "id":"1415228280000_Asgrad-Rolling-Upgrade-Log_AutoScalingGroupName:testworkload-r01_CPUUtilization",
      "timestamp":1415228280000,
      "timestmapFormatted":"2014-11-06T09:58:00+1100",
      "label":"CPUUtilization",
      "dimension":"AutoScalingGroupName:testworkload-r01",
      "logOrigin":"Asgrad-Rolling-Upgrade-Log",
      "logs":[
         "2014-11-06_09:58:02 Updating launch from testworkload-r01-20140522141935 with ami-4583197f into testworkload-r01-20
         "2014-11-06_09:58:02 Started on thread Task:Pushing ami-4583197f into group testworkload-r01 for app testworkload.",
         "2014-11-06_09:58:03 Create Launch Configuration 'testworkload-r01-20141106095802' with image 'ami-4583197f'",
         "2014-11-06_09:58:03 Updating group testworkload-r01 to use launch config testworkload-r01-20141106095802",
         "2014-11-06_09:58:03 Update Autoscaling Group 'testworkload-r01'",
         "2014-11-06_09:58:04 The group testworkload-r01 has 8 instances. 8 will be replaced, 2 at a time.",
         "2014-11-06_09:58:04 Remove instances [i-41a02f8e] from Load Balancer ELB-01",
         "2014-11-06_09:58:04 Disabling testworkload / i-41a02f8e in 1 ELBs.",
         "2014-11-06_09:58:04 Sorted testworkload instances in testworkload-r01 by launch time with oldest first",
         "2014-11-06_09:58:05 Disabling testworkload / i-46a02f89 in 1 ELBs."
      ],
      "actualValue":19.80,
      "estimatedValue":18.498,
      "estimatedValueDiscreted":18,
      "anomalyStatus":false,
      "anomalyStatusConfidence":"NotApplicable"
   }
},
{
   "_index":"pod",
   "_type":"anomalyReport",
   "_id":"1415228820000_Asgrad-Rolling-Upgrade-Log_AutoScalingGroupName:testworkload-r01_TerminatedInstance",
   "_score":1,
   "_source":{
      "id":"1415228820000_Asgrad-Rolling-Upgrade-Log_AutoScalingGroupName:testworkload-r01_TerminatedInstance",
      "timestamp":1415228820000,
      "timestmapFormatted":"2014-11-06T10:07:00+1100",
      "label":"TerminatedInstance",
      "dimension":"AutoScalingGroupName:testworkload-r01",
      "logOrigin":"Asgrad-Rolling-Upgrade-Log",
      "logs":[
         "2014-11-06_10:07:02 Terminating instance i-258982ea",
         "2014-11-06_10:07:02 Terminate 1 instance [i-258982ea]",
         "2014-11-06_10:07:02 Waiting up to 1h 10m for new instance of testworkload-r01 to become Pending.",
         "2014-11-06_10:07:20 It took 2m 49s for instance i-238982ec to terminate and be replaced by i-318e85fe",
         "2014-11-06_10:07:20 Waiting up to 50m for Pending i-318e85fe to go InService.",
         "2014-11-06_10:07:23 Instance of testworkload on i-318e85fe is in lifecycle state Pending",
         "2014-11-06_10:07:51 It took 31s for instance i-318e85fe to go from Pending to InService",
         "2014-11-06_10:07:55 Waiting 30 seconds for testworkload on i-318e85fe to be ready."
      ],
      "actualValue":1,
      "estimatedValue":0.051,
      "estimatedValueDiscreted":0,
      "anomalyStatus":true,
      "anomalyStatusConfidence":"NotApplicable"
   }
},
```


Figure 4.3: A sample of an Anomaly Report

access the monitoring data.

Next, in our Anomaly-Checker, we aimed to make the anomaly checking solution capable of being used at both run-time and for post-mortem analysis. Therefore, we implemented a service so that Anomaly-Checker can be called, based on the given time intervals and the duration needed for Anomaly-Checker.

The main service for anomaly detection performs anomaly checking for one or multiple monitoring metrics based on the following parameters:

- *logOrigin*: Every monitoring operation should be identified with an iden-

tifier (an ID assigned to the project that logs are obtained from). Here *logorigin* refers to the identifier of the operation.

- *dimension*: Dimension property refers to the node or source of metric observation, such as the instance ID of a virtual machine to monitor.
- *labels*: Metric names can be more than one metric, such as CPUUtilization, and TerminatedInstance.
- *startTime*: The data from a given timestamp will be processed for anomaly checking. The start time can be set as a current time for real-time anomaly checking or as a time in the past for post-mortem anomaly checking.
- *endTime*: This indicates the end time of anomaly checking. A null value will cause the anomaly checking to be continued endlessly.
- *interval*: This indicates the interval of anomaly checking in milliseconds.

As can be seen in the activity diagram depicted Fig. 4.4, this AnomalyChecker service receives the above inputs, and, based on the coefficient values of the assertion equation that is given in a configuration file, it will start to calculate the predicted value of each metric. Once the predicted value is obtained, it is compared to the actual value, and if the prediction is not close enough to the actual value, it will update the anomaly status to true, meaning an anomaly is detected for a given timestamp. The high-level steps of the process of anomaly checking from a service call to return an anomaly report is shown in Fig. 4.4.

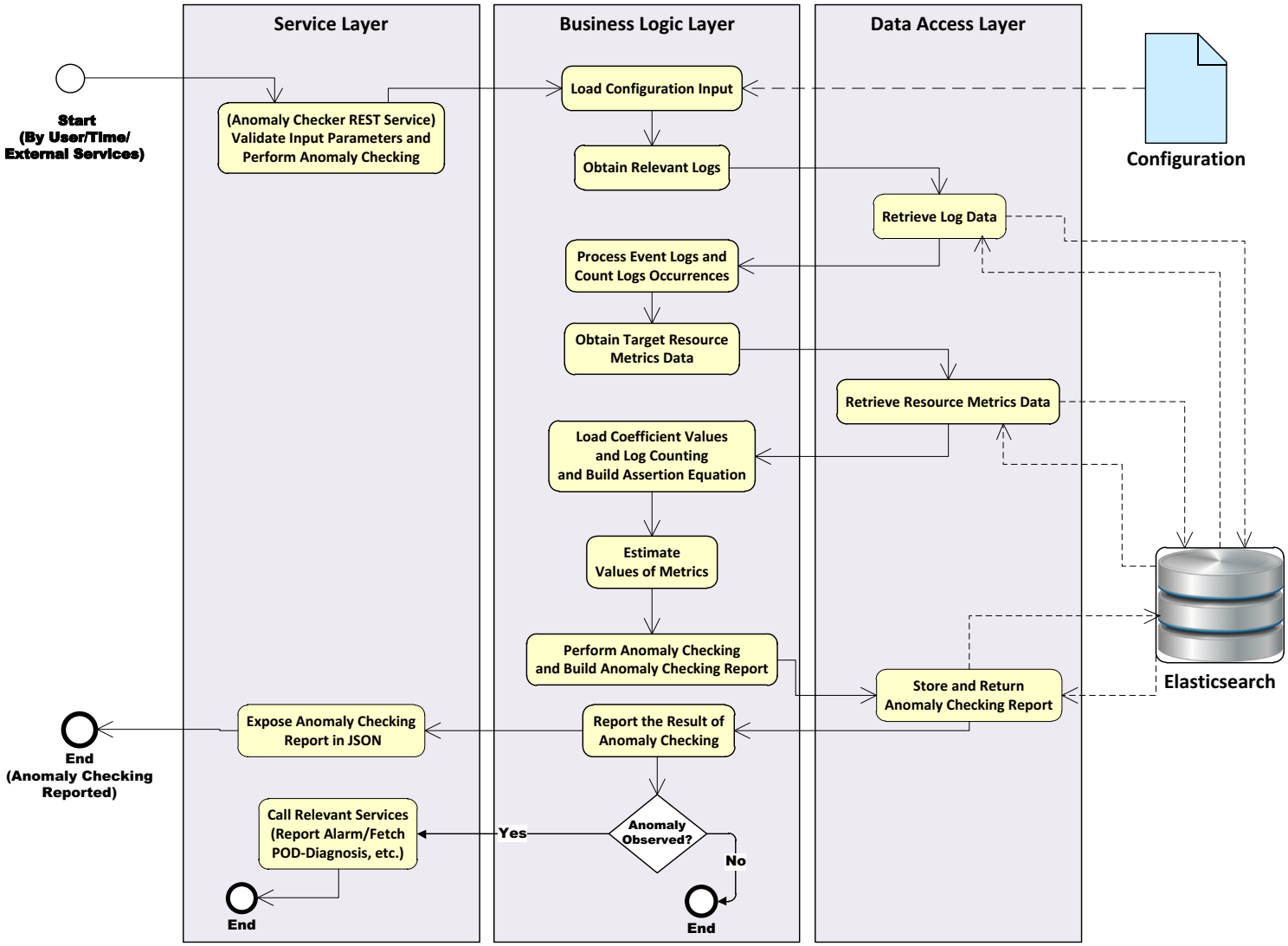


Figure 4.4: Anomaly Checking Activity Diagram.

Fig. 4.3 shows a sample of two records of anomaly checking (as described for Anomaly Report schema in the previous section) of a rolling upgrade case study. The first record shows the checking of anomaly detection for an aggregated CPU utilization metric of a scaling group (*dimensions*), and the log events that are reported at that timestamp. The anomaly status (*anomalyStatus*) as the result of comparing actual value and estimated value is *false* in this record, meaning that no anomaly was observed in this timestamp for this metric. The next record (in the brackets in the bottom block) reports the anomaly status as *true*, checking *TerminatedInstance* for the scaling group of a different timestamp.

As we mentioned earlier in this chapter, the focus of the anomaly checking prototype that we designed and implemented for this thesis was on the detection of anomalies only. The role of visualisation of monitoring data is designed to be handled by POD-Viz [130] in the future. Also, the role of anomaly debugging and diagnosis is planned to be integrated with POD-Diagnosis [140].

4.5 Summary

In this chapter, we presented an overview of the anomaly checking prototype that is used to conduct experiments and evaluate the proposed approach of this thesis.

Some of the main components that have been implemented as part of the development of this prototype to conduct our experiments include log-preprocessing for validation of logs and log sanitisation, conversion and persistence of logs to log events according to the log event schema of Elasticsearch, conversion and storage of raw monitoring metric data in Elasticsearch, processing and extracting the interpolated occurrence strength of each log event, processing and extracting indirect monitoring metrics, and finally, the anomaly checking for processing data to detect the occurrence of anomalies from the analysis of both logs and metrics. We used this prototype as part of our investigation and experimentation with two case studies that we will discuss in detail in the next two chapters.

Chapter 5

Rolling Upgrade Case Study

In this chapter, we will present a case study to explore the effectiveness of the proposed approach of this thesis that was proposed in Chapter 3. For this purpose, we will use the tool described in the previous chapter. Specifically, we are looking to evaluate to what extent, with the help of the proposed approach, we can address the research questions of this thesis that were articulated in Chapter 1, page 5. Related to our research questions we have the following hypotheses:

- By employing the techniques introduced in Section 3.3 using regular expressions, representing logs as quantitative metrics using interpolated occurrence strength of log event type, and cluster log events for a set of log activities, we may be able to map log activities to the observed resource metrics (addressing RQ1).

- By using the statistical analysis explained in Section 3.4, we may be able to identify resource metrics that are affected by the system behaviour represented in logs, and also to identify the most sensitive metrics that would be the best candidates for anomaly detection (addressing RQ2).

- By exploiting the predictive and exploratory power of regression analysis of log activities as input variables and resource metrics as target variables, we may be able to build a statistically supported model to derive assertions that enable high-accuracy, unsupervised, contextual anomaly detection (addressing RQ3).

In this chapter, we will investigate the above hypotheses with employing Rolling Upgrade operation as our case study.[1]

5.1 Rolling Upgrade

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[2] on top of Amazon Elastic Computing Cloud (EC2), as a case study of such an operation.

A rolling upgrade operation is an excellent example of a sporadic cloud operation . Applications in the cloud are deployed on a collection of virtual machines (VMs). As one common approach to upgrade process, 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 a newly prepared image through an upgrade process, such as rolling upgrade [9].

A rolling upgrade replaces VM instances, x at a time [129], e.g. upgrading 400 instances in total by upgrading 10 instances concurrently at any given time during the operation. In Asgard – a Netflix cloud deployment tool – the rolling upgrade process has the following steps - also shown in Fig. 5.1:

1. Create a new Launch Configuration (LC[3]),
2. Update the Auto Scaling Group (ASG[4]),
3. Set user-specified rolling policy,
4. Remove and deregister the instance from Elastic Load Balancer (ELB[5]),

[1] Parts of this chapter have been published: [36, 37].
[2] https://github.com/Netflix/asgard
[3] A launch configuration is a template specifying the information for launching virtual machines instances in an Auto Scaling Group [6].
[4] "An Auto Scaling group contains a collection of EC2 instances that share similar characteristics and are treated as a logical grouping for the purposes of instance scaling and management" [6].
[5] "Elastic Load Balancing distributes incoming application traffic across multiple EC2 instances, in multiple Availability Zones"[6].

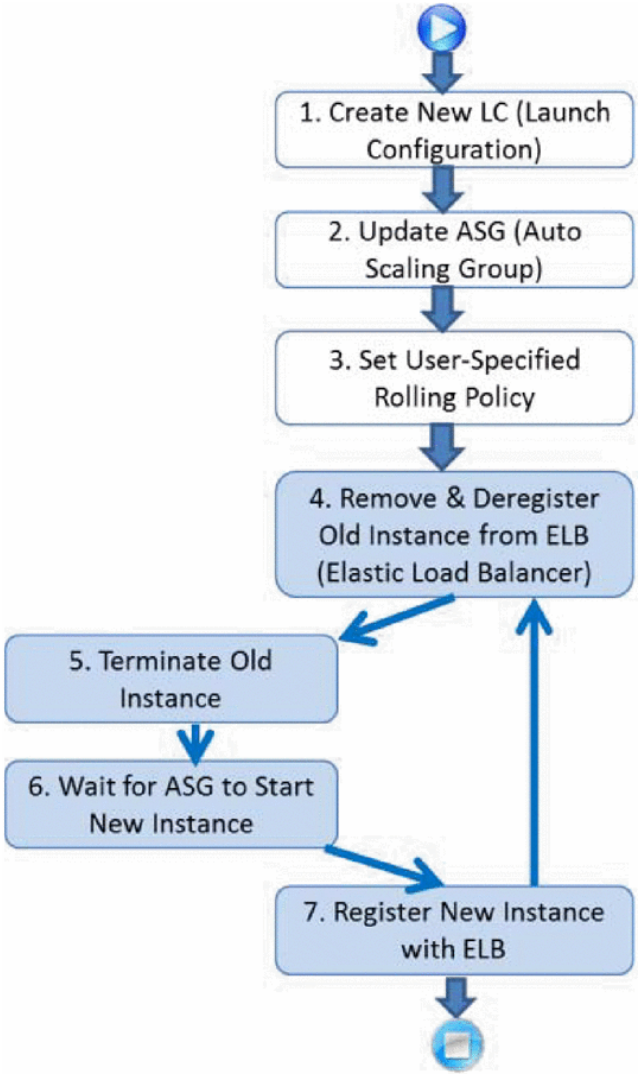


Figure 5.1: Flow Chart of a Rolling Upgrade Process [42].

5. Terminate the old instance,

6. Wait until the auto-scaling group replaces the missing instance with a new instance (running the updated version of the application),

7. The new instance is registered with the ELB.

8. Repeat the steps 4 to 7 for each VM instance.

In the process of rolling upgrade, there is a chance that an instance faces a configuration change or an error at any time during these steps. For example, a VM freezes, crashes and/or it becomes unresponsive. Such failures may be caused by a problem stemming from the resources the VM is running on; memory over-usage due to an increased system load; an application bug that

stresses the VM; an operating system kernel bug; or through random system termination for assessing of systems resiliency and recoverability in production. The occurrence of any of these failures during an upgrade operation can put the upgrade process on hold or even derail it.

In a nutshell, rolling upgrade is an excellent exemplar operation for our investigation for the following reasons: upgrade operations are one of the most error-prone operations [34]; the rolling upgrade operation is one of the most 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.

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 a few times a week [39] to many times per day [88], updating hundreds of machines, without causing any service downtime. Such frequency of executing this sensitive operation was another reason that inspired us to choose this case study for investigation in this thesis.

For further information about rolling upgrade, readers can refer to [78], [132], or [108].

5.1.1 Experimental Environment

In this section, we give an overview of the experimental environment. 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 widespread 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.

One concern of our study was to choose a suitable cluster size for the experiments that resemble a realistic scenario in the industry. While there are some studies [21, 116, 133] and guidelines [45, 122] regarding optimal virtual

machines placement and configuration setting for server architecture, to the best of our knowledge, there is no survey report or statistics available on server size in industry. Hence, based on our intuition, we chose two different configuration settings that emulate both a medium-scale server and a fairly large-scale server environment.

In this direction, we performed our analysis based on two separate case studies of rolling upgrades: first, to emulate a medium-size cluster size, we conducted multiple runs of rolling upgrade of 8 instances ,upgrading 2 instances at a time; and the second case study, to emulate a fairly large-scale cluster, we conducted rolling upgrade of 40 instances, upgrading 4 instances at a time.

We conducted a total of 20 rounds of rolling upgrade operations, for each case study running 10 rounds of rolling upgrade on Amazon EC2. In this process we collected logs from our operation execution environment through Netflix Asgard, and we gathered monitoring metric data from Amazon CloudWatch. The above configuration gave us a fairly good representation of a medium and a large-scale server environment.

5.1.2 Netflix Asgard - Event Logs from Operations Tools

In our case study, we used Netflix Asgard to execute rolling upgrade operations and to collect operation logs. Asgard has been one of the primary tools for application deployment and cloud management at Netflix for years [112]. Asgard is an open-source web-based tool published by Netflix for managing cloud-based applications and infrastructure. Asgard automates some of the AWS cloud operations, such as deployment and upgrade.

Asgard was developed by Netflix, one of AWS's largest customers, to provide a higher-level management interface, and since it has been released publicly, is in widespread use. Its log fulfils our base assumption, and contains high-quality textual messages – albeit the latter is not required in our approach. A log event should have a timestamp and a description in order to be employed by our approach; this format is broadly accepted in industry [29, 54, 95, 119].

The rolling upgrade process can be observed by tracking the log events of

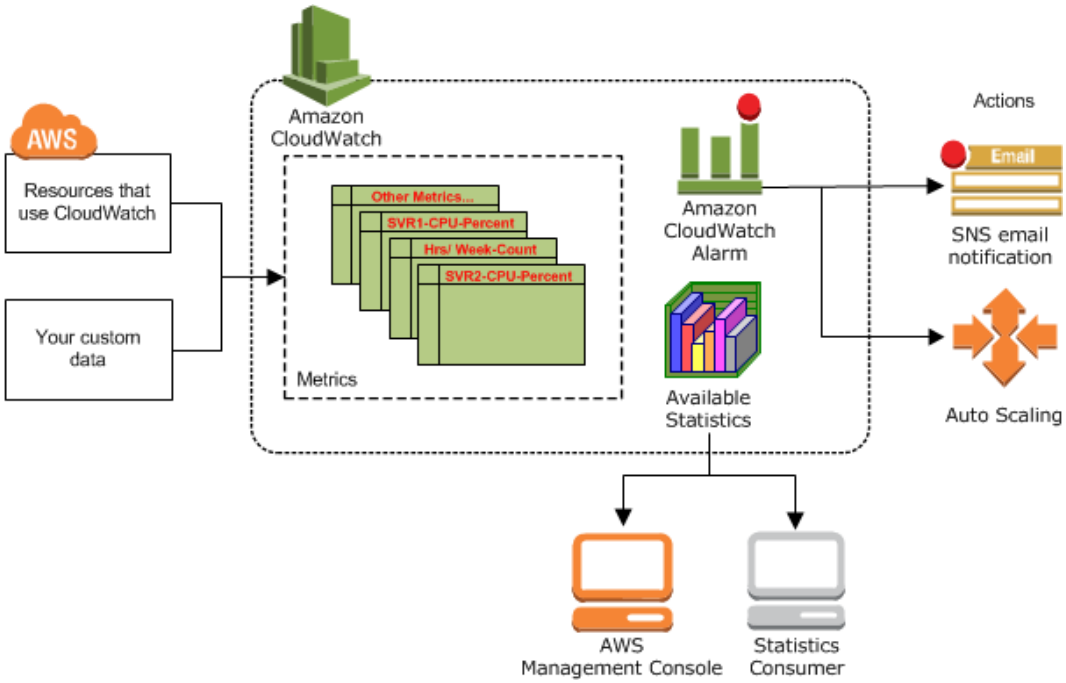


Figure 5.2: CloudWatch Overview - source: AWS documentation.

rolling upgrade operation with Asgard administration console. As was previously highlighted, we ran 20 rounds of rolling upgrade operations. For each round of rolling upgrade operation, we collected operation log events from Asgard as the contextual source of information for monitoring and anomaly detection throughout this chapter. Having the operation logs along with metrics data give us the data needed to evaluate our approach.

5.1.3 Amazon CloudWatch - Resources Metrics

Metrics are data about the state or performance of systems [6]. In AWS, a metric represents a time-ordered set of data points that are published to CloudWatch. In other words, a metric is a variable to monitor, and the data points represent the values of that variable over time [6].

CloudWatch is, in essence, a metrics repository. It provides monitoring metrics for many Amazon services, including EC2, ASG, and ELB. In addition to the above, CloudWatch provides a feature to retrieve simple statistics about data, and define rules for alarm notification, as well as being a console for configuration and metric visualisation. Fig. 5.2 shows an overview of Amazon CloudWatch.

Metric data is available in five-minute intervals by default, but CloudWatch can be configured, at extra cost, to collect metrics as precisely as at one-minute intervals. [6] The data is available in JSON file format and can be retrieved through an API. A sample of CloudWatch metric data in JSON format is shown in Fig. 5.3. Following describes the elements of CLoudWatch JSON metric schema:

- **Label**: Metric Name e.g. CPUUtilization,
- **Dimensions**: Node or a name/value pair that uniquely identifies a metric such as ASG, ELB and individual VM instances,
- **Datapoints**
  - **timestamp**: the timestamp of the observation ,
  - **sampleCount**: number of data point collection within the observation period - e.g. 8 times in one minute,
  - **average**: The value of Sum / SampleCount during the specified period,
  - **sum**: All values submitted for the matching metric added together, e.g. all bytes received from Network,
  - **minimum**: The lowest value observed during the specified period.,
  - **maximum**: The highest value observed during the specifie d period.,
  - **unit**: The unit of measure, such as Bytes, Seconds, Count, and Percent.

By default, several metrics related to individual VM instances can be collected by CloudWatch, including CPU utilization, network traffic (incoming/outgoing), failed health checks, and so on. For a whole ASG, averages can be obtained. Fig. 5.3 shows two sample monitoring record at the same

[6] CloudWatch retains metrics data as follows: "Data points with a period of 60 seconds (1 minute) are available for 15 days; Data points with a period of 300 seconds (5 minutes) are available for 63 days; Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)" [6].

timestamp, the top one indicates CPUUtilization for an AutoScalingGroup, and the bottom one shows the network input (NetwrokIn) for a specific VM instance. In the samples, the average refers to the mean of data points collected within a time window. For example, as shown in Fig. 5.3, *sampleCount* of *CPUUtilization* shows *eight* times CPU utilisations have been recorded, where the average of these *eight* times was 37.96 percent, the minimum 35 percent and the maximum 42.62 percent.

These metric data can be used for purposes like monitoring the health of the system or for custom auto-scaling. In this study, we just have the focus on employing the data for system health and performance monitoring.


```
[
  {
    "Label": "CPUUtilization",
    "Dimensions": "AutoScalingGroupName:testworkload-r01",
    "Datapoints": [
      {
        "timestamp": "Nov 6, 2014 12:53:00 AM",
        "sampleCount": 8,
        "average": 37.96,
        "sum": 303.68,
        "minimum": 35,
        "maximum": 42.62,
        "unit": "Percent"
      }
    ]
  },
  {
    "Label": "NetworkIn",
    "Dimensions": "InstanceId:i-b85b5b77",
    "Datapoints": [
      {
        "timestamp": "Nov 6, 2014 12:53:00 AM",
        "sampleCount": 1,
        "average": 37597,
        "sum": 37597,
        "minimum": 37597,
        "maximum": 37597,
        "unit": "Bytes"
      }
    ]
  }
```


Figure 5.3: A sample JSON output of two different metrics of CloudWatch.

5.1.4 Direct and Derived Metrics

In our study, to verify the effect of logged events, we are not just interested in the metrics that show the resource consumption of resources such as memory usage and network traffic, we are also interested in having metrics that represent the transitions between states of a VM instance such as the time a VM goes from launch to pending. Therefore, we are interested in both *performance* and *state-based* metrics.

Direct metrics are basic metrics that are often available by default with monitoring tools and available for most of the metrics that show the capacity or resource consumption of various resources. These types of metrics are the metrics that amount or the percentage or the degree of the changes of the usage of the resources, such as CPU utilisation, disk read IO, memory usage, network traffic, number of SQL threads running, and number of open connections.

Derived metrics are the metrics that may not always be available directly and they often indicate the transition of the status of resources from one state to another. One important example of these types of metrics are the ones indicating the status of a VM. For example, CloudWatch (AWS monitoring service), includes no metric that explicitly shows the number of instances started or terminated within an auto scaling group.

As rolling upgrade operation directly affect the life cycle of virtual machine instances, in our experiment, we were interested to know the state of VM instances at each monitoring time interval. However, we noticed this metric type was not directly available in CloudWatch. The state-based metrics are necessary metrics because the process of rolling upgrade with baked (prepared) images involves taking a running virtual machine out of service, terminating the machine, and replacing it with a new one, and, finally, registering the new machine as an in-service machine. The life-cycle of state changes of Amazon EC2 VMs is shown in Fig. 5.4. For example, once a termination action is triggered through operation execution, one EC2 VM instance should transition from state “running” to “shutting down” and eventually to state “terminated”.

In CloudWatch, there is a metric indicating the total number of healthy machines; however, this is only partly indicative of the number of started or

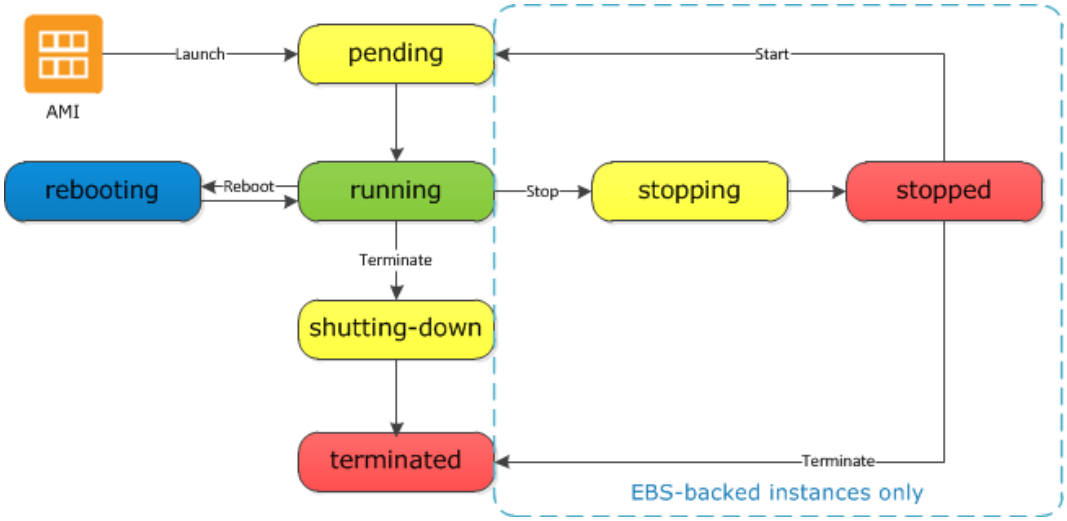


Figure 5.4: Amazon EC2 instance lifecycle- source: AWS documentation.

terminated machines: if, during any minute, a new machine becomes active and an old one is terminated directly after, the total number of healthy machines remains static. This is a very common occurrence during a rolling upgrade. To address this, we implemented a mechanism to derive the number of terminated VM instances and the number of started instances for each minute by tracking the individual metrics of each VM instance.

In CloudWatch, the IDs of all VM instances are available, and it is a straightforward task to understand whether a machine is in service or not by querying its metrics from CloudWatch API. Given that t denotes the observation time and v denotes the individual in-service VM instances, and supposed we have:

$$\begin{cases} 0 & \text{if VM X is not in service during time t} \\ 1 & \text{if VM X is in service during time t} \end{cases}$$

then, the total number of in-service instances in time t is represented by capital I and can be obtained as:

$$I_t = \sum v1_t + v2_t + ...vN_t$$

When there is an absence of the record of active metrics for an instance ID in the previous minute, we know this is a new machine. Given this description,

we define the started instances as those that have $vX_{t-1} = 0$ and $vX_t = 1$, and the opposite for terminated machines.

We denote the new launched VM instances as sv and represent the total number of new launched instances with SI (Started Instance), and obtain the metric of *SI (StartedInstance)*:

$$SI_t = \sum sv1_t + sv2_t + ...svN_t$$

By having the information of total active VM instances of previous minute and current minute, and the number of newly launched VM instances, we obtain the *TI (TerminatedInstance)* of time t as below:

$$TI_t = \left|(\sum I_t - \sum SI_t) - \sum I_{t-1}\right|$$

Based on the above, we derived precise metrics for the numbers of started and terminated instances, respectively. These metrics derived from CloudWatch data are a cornerstone for the successful application of our approach to rolling upgrade.

It is worth noting that many metrics may be available in CloudWatch, both at the instance level and at the group level. Considering instance-level metrics monitoring are, 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 the above-mentioned experiments, including the metrics of started instances and terminated instances.

5.2 Log Analysis

5.2.1 Log Parsing and Log Representation as a Quantitative Metric

In order to map metrics obtained from CloudWatch to event logs from Asgard, we extract a set of metrics that show the occurrences of different event logs. In programs with any form of repetition, it is common to have logged events of recurring event types. In our case study of rolling upgrade, every time a VM is terminated, the same type of event is logged, where only certain parameters (VM ID, timestamp, etc.) differ.

For processing the raw event logs of Asgard, we followed the steps explained in Section 3.3.1, by which all unique types of event logs for an operation are identified and corresponding regular expressions are extracted. In this process, first, the timestamps and log description are extracted for each log line. Then, the log description is tokenized, and for each token that does not contain digits, the exact string of the token is extracted as the regular expression.

For example, a string of "Create Launch Configuration" in a log is presented in a regular expression as "*\sCreate\sLaunch\sConfiguration*" [7]. For those tokens with digits such as IP address, VM instance IDs or a reported waiting

[7] \s matches whitespaces, tabs and new lines.

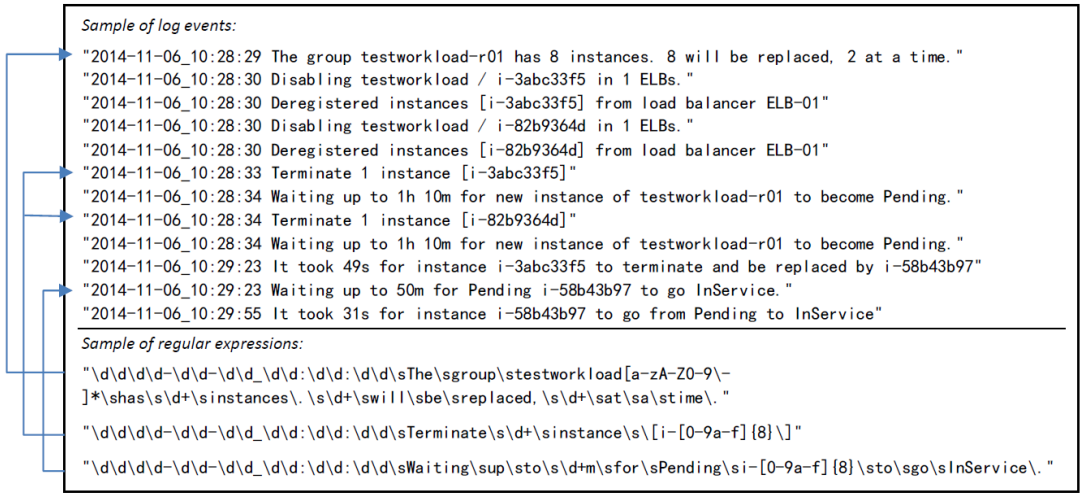


Figure 5.5: Sample of log event and extracted regular expression of the rolling upgrade operation with Netflix Asgard.

time, the regular expression that matches any digits with the format of the token is extracted. For instance, the extracted pattern for a token string like *"instance i-4583197f"* which indicates a VM instance ID is as *"instance\s\[i-[0-9a-f]8\]"*. The output of the above step is a set of regular expression patterns that represent unique types of event logs. Fig. 5.5 shows an example of the extracted regular expression pattern for the given log lines.

The output of the above step gave us 18 event types for our rolling upgrade case study using Asgard. Visualization of the occurrence of event log types based on the above may help with recognizing patterns in system operations, and may help better understand the execution of an operation. In this direction, Fig. 5.6 shows the pattern of occurrences of the event logs throughout the process of rolling upgrade operation for updating four virtual machine instances.

In the log occurrence pattern shown in Fig. 5.6, the axis that is labelled as *Logged event types* lists event type 1 (ET01) to event type 18 (ET18); the axis labeled as *Timestamp* shows the timeline of the operation; and the vertical axis shows the number of occurrences of each logged event type for each minute of the operation. Looking at the figure, it reveals that there are certain patterns in the event type occurrences. For instance, ET01, ET02, ET04, ET05, ET06, ET07 and ET08 appear just at the beginning of the operation, which may imply that they are related to the logs associated with the starting and preparation steps of the operation. In contrast, there is a reoccurring pattern for event types ET09 to ET17, which may represent recurring activities such as VM termination. In this chapter, 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.

Amazon CloudWatch offers metrics with a granularity no finer than one minute. In contrast, events can be logged whenever they occur with no regulation and the intervals can be from split seconds to minutes. Therefore, the log and metric data need to be mapped. Since we can observe actual changes to

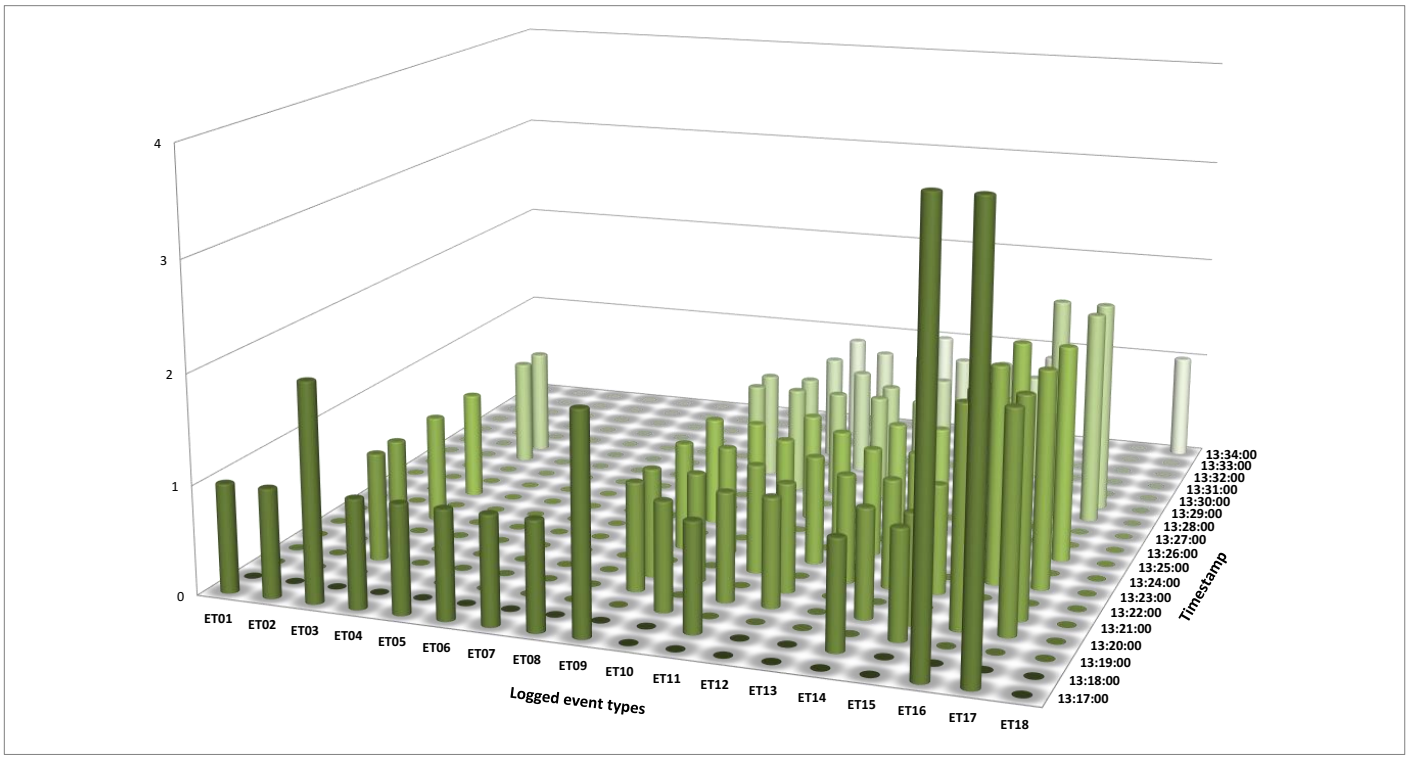


Figure 5.6: Visualization of occurrence of 18 event types of rolling upgrade for 4 VM instances.

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 time window to the respective minute. This gives us the interpolated occurrence strength of each event type. It is worth noting that, in the beginning, we started our investigation based on the simple counting of occurrence of log event types without interpolated values, but this didn't lead to sensible clustering due to the collinearity observed among event types, therefore, we incorporated interpolated occurrence strength into our analysis.

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 a set of log events happening. 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. For instance, say event *E1* happened at t minutes and 30 seconds, then its occurrence strength is counted as 0.5 for

both minute t and minute $t+1$. If it happened at t minutes and 15 seconds, the occurrence strength for minute t is 0.75 and for minute $t+1$ it is 0.25. This interpolated occurrence strength allows us to map the event log data onto the one-minute interval of our cloud metric data.

5.2.2 Log Events Correlation Clustering - Mapping Low Granular Logs to a Set of Activities

In the previous section, we derived a metric that represents the interpolated occurrence strength of log event types. In this section, we employ a Pearson correlation analysis to find the log event types which are highly correlated with their occurrence. Therefore, we generated a Pearson correlation coefficient for two different data sets of upgrading 8 machines and 40 machines.

First, we generated correlation data for running a rolling upgrade of eight virtual machine instances, upgrading two instances at a time. Then, we defined a rule that event types be grouped together when they had a correlation strength of more than 75% (Pearson-r > 0.75), where 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 every other event type of the group that formed an activity. The generated correlation matrix(using SPSS) including the value of Pearson Correlation (Pearson-r) as well statistical significance (*p-value*) of log event types is shown in Fig. 5.7. Also, Fig. 5.8 shows a generated radar graph based on these correlation values. Considering the correlation strength of 0.75, the graph clearly shows the close correlation mapping of E01-E07, E08-E11, E12-E14, E15-E16, E17, and E18.

One may choose a higher or lower level, depending on the desired abstraction level to be obtained from logs. We chose Pearson-r > 0.75, as it is low enough to avoid multicollinearity in our regression analysis and high enough to associate strongly correlated event types together. To make sure that the correlation of activities is not affected by different configurations 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.

Correlations

		ET01	ET02	ET03	ET04	ET05	ET06	ET07	ET08	ET09	ET10	ET11	ET12	ET13	ET14	ET15	ET16	ET17	ET18
ET01	Pearson Correlation	1	1	0.928	0.923	0.923	0.911	0.911	0.26	0.26	0.15	0.148	-.063	-.060	-.063	-.073	-.073	-.073	-.019
	Sig. (2-tailed)		0.000	.000	.000	.000	.000	.000	.000	.000	.001	.001	.153	.176	.153	.100	.099	.099	.672
ET02	Pearson Correlation	1	1	0.928	0.924	0.924	0.912	0.912	0.261	0.261	0.152	0.15	-.063	-.060	-.063	-.073	-.073	-.073	-.019
	Sig. (2-tailed)	0.000		.000	.000	.000	.000	.000	.000	.000	.001	.001	.154	.177	.154	.100	.099	.100	.673
ET03	Pearson Correlation	0.928	0.928	1	1	1	0.997	0.997	0.293	0.292	0.198	0.195	-.064	-.061	-.064	-.074	-.074	-.074	-.019
	Sig. (2-tailed)	.000	.000		0.000	0.000	0.000	0.000	.000	.000	.000	.000	.146	.169	.146	.094	.093	.093	.667
ET04	Pearson Correlation	0.923	0.924	1	1	1	0.998	0.998	0.293	0.293	0.201	0.198	-.064	-.061	-.064	-.074	-.074	-.074	-.019
	Sig. (2-tailed)	.000	.000	0.000		0.000	0.000	0.000	.000	.000	.000	.000	.147	.171	.147	.095	.094	.095	.668
ET05	Pearson Correlation	0.923	0.924	1	1	1	0.998	0.998	0.293	0.293	0.201	0.198	-.064	-.061	-.064	-.074	-.074	-.074	-.019
	Sig. (2-tailed)	.000	.000	0.000	0.000		0.000	0.000	.000	.000	.000	.000	.147	.171	.147	.095	.094	.095	.668
ET06	Pearson Correlation	0.911	0.912	0.997	0.998	0.998	1	1	0.296	0.295	0.209	0.206	-.063	-.060	-.063	-.073	-.073	-.073	-.019
	Sig. (2-tailed)	.000	.000	0.000	0.000	0.000		0.000	.000	.000	.000	.000	.152	.176	.152	.099	.098	.098	.672
ET07	Pearson Correlation	0.911	0.912	0.997	0.998	0.998	1	1	0.296	0.295	0.209	0.206	-.063	-.060	-.063	-.073	-.073	-.073	-.019
	Sig. (2-tailed)	.000	.000	0.000	0.000	0.000	0.000		.000	.000	.000	.000	.152	.176	.152	.099	.098	.098	.672
ET08	Pearson Correlation	0.26	0.261	0.293	0.293	0.293	0.296	0.296	1	1	0.777	0.77	-0.247	-0.234	-0.247	-0.106	0.126	0.627	-.073
	Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000		0.000	.000	.000	.000	.000	.000	.017	.004	.000	.097
ET09	Pearson Correlation	0.26	0.261	0.292	0.293	0.293	0.295	0.295	1	1	0.778	0.771	-0.247	-0.234	-0.247	-0.106	0.125	0.627	-.073
	Sig. (2-tailed)	.000	.000	.000	.000	.000	.000	.000	0.000		.000	.000	.000	.000	.000	.016	.004	.000	.097
ET10	Pearson Correlation	0.15	0.152	0.198	0.201	0.201	0.209	0.209	0.777	0.778	1	0.998	-0.25	-0.236	-0.25	-0.213	-0.111	0.637	-.074
	Sig. (2-tailed)	.001	.001	.000	.000	.000	.000	.000	.000	.000		0.000	.000	.000	.000	.000	.011	.000	.094
ET11	Pearson Correlation	0.148	0.15	0.195	0.198	0.198	0.206	0.206	0.77	0.771	0.998	1	-0.251	-0.237	-0.251	-0.216	-0.119	0.631	-.074
	Sig. (2-tailed)	.001	.001	.000	.000	.000	.000	.000	.000	.000	0.000		.000	.000	.000	.000	.007	.000	.092
ET12	Pearson Correlation	-.063	-.063	-.064	-.064	-.064	-.063	-.063	-0.247	-0.247	-0.25	-0.251	1	0.751	1	.043	-0.141	-0.25	-.064
	Sig. (2-tailed)	.153	.154	.146	.147	.147	.152	.152	.000	.000	.000	.000		.000	0.000	.331	.001	.000	.146
ET13	Pearson Correlation	-.060	-.060	-.061	-.061	-.061	-.060	-.060	-0.234	-0.234	-0.236	-0.237	0.751	1	0.751	0.354	0.102	-0.237	-.061
	Sig. (2-tailed)	.176	.177	.169	.171	.171	.176	.176	.000	.000	.000	.000	.000		.000	.000	.021	.000	.169
ET14	Pearson Correlation	-.063	-.063	-.064	-.064	-.064	-.063	-.063	-0.247	-0.247	-0.25	-0.251	1	0.756	1	.043	-0.141	-0.25	-.064
	Sig. (2-tailed)	.153	.154	.146	.147	.147	.152	.152	.000	.000	.000	.000	0.000	.000		.331	.001	.000	.146
ET15	Pearson Correlation	-.073	-.073	-.074	-.074	-.074	-.073	-.073	-0.106	-0.106	-0.213	-0.216	.043	0.354	.043	1	0.813	-.081	-.074
	Sig. (2-tailed)	.100	.100	.094	.095	.095	.099	.099	.017	.016	.000	.000	.331	.000	.331		.000	.067	.094
ET16	Pearson Correlation	-.073	-.073	-.074	-.074	-.074	-.073	-.073	0.126	0.125	-0.111	-0.119	-0.141	0.102	-0.141	0.813	1	0.172	-.068
	Sig. (2-tailed)	.099	.099	.093	.094	.094	.098	.098	.004	.004	.011	.007	.001	.021	.001	.000		.000	.123
ET17	Pearson Correlation	-.073	-.073	-.074	-.074	-.074	-.073	-.073	0.627	0.627	0.637	0.631	-0.25	-0.237	-0.25	-.081	0.172	1	0.265
	Sig. (2-tailed)	.099	.100	.093	.095	.095	.098	.098	.000	.000	.000	.000	.000	.000	.000	.067	.000		.000
ET18	Pearson Correlation	-.019	-.019	-.019	-.019	-.019	-.019	-.019	-.073	-.073	-.074	-.074	-.064	-.061	-.064	-.074	-.068	0.265	1
	Sig. (2-tailed)	.672	.673	.667	.668	.668	.672	.672	.097	.097	.094	.092	.146	.169	.146	.094	.123	.000	

Figure 5.7: Correlation matrix generated by SPSS based on interpolated occurrence strength of each event type.

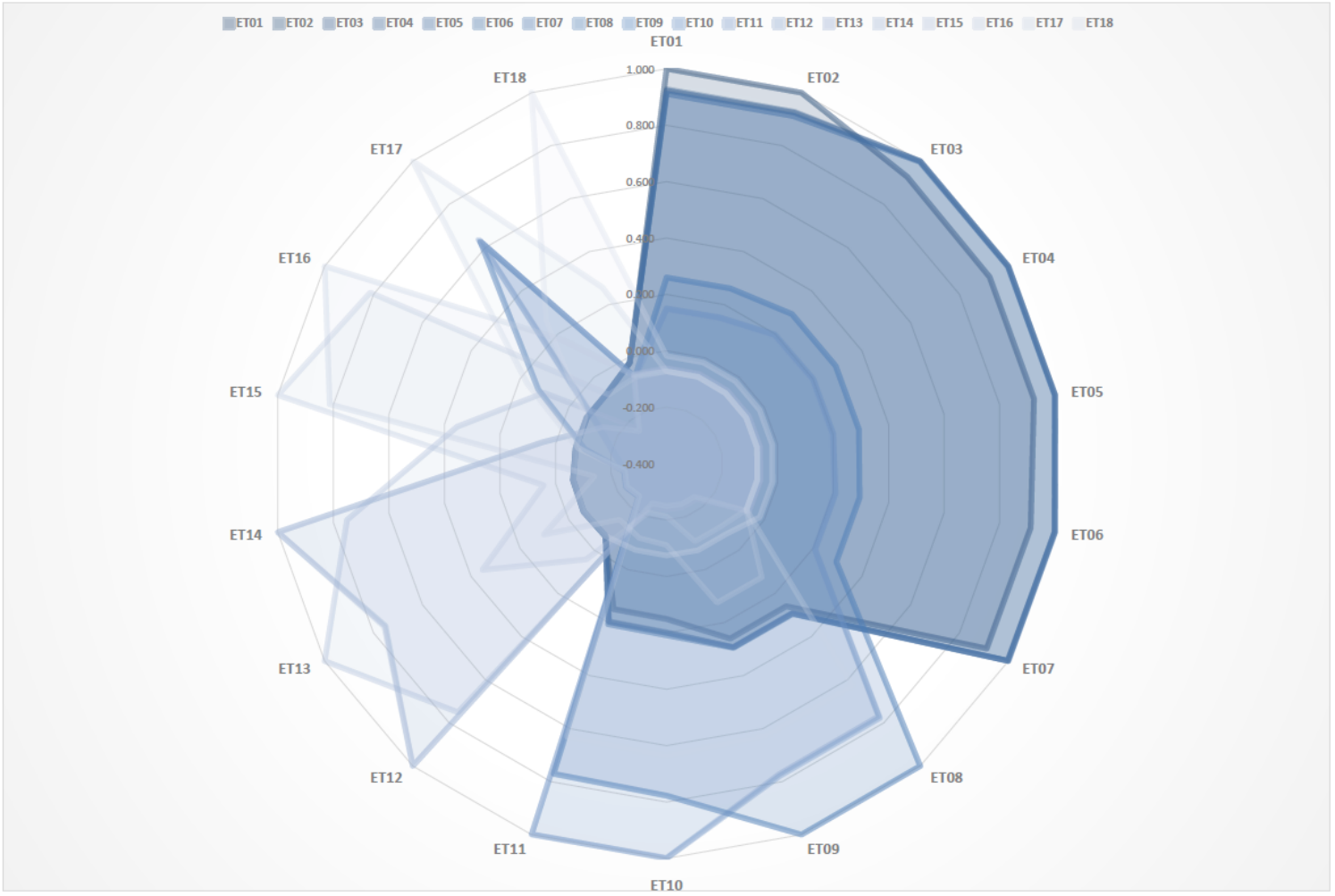


Figure 5.8: Correlation clustering graph based on values given in Fig.5.7.

Table 5.1: 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 (Launch Configuration for Auto Scaling Group)
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

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 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 5.1, shows that all the event types of each cluster are meaningfully related to each other. In other words, individual event types are clustered to activities that fulfil a particular goal such as Launch Configuration or VM Termination. 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 the sake of the presentation of our analysis, each cluster was given a name according to the context of its event types. The full log to activity mapping is given in Table 5.1.

5.3 Metric Selection

To investigate the relationship between the occurrence of event types and cloud metrics in the regression model, we assigned the activities derived from log clustering as *predictor variables* and the 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.

5.3.1 Log-Metric Correlation Learning - Which Metrics Should be Selected for Monitoring?

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 eight and 40 instances. The results of regression analysis over group-level monitoring metrics are shown in Table 5.2 (metrics with the highest correlation are shown in bold).

In the table, R denotes the correlation between the occurrences of activities extracted from the event logs and a monitoring metric, and R^2 indicates how well the model from activites predicts values of the target metric. In general, R^2 is used to assess the predictive power of the regression model for the given predictors and target variables [99].

$Adj.R^2$ is a modification version of R^2 that adjusts for the number of predictors in a model [99]. 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 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 $Adj.R^2$ [89].

Table 5.2: Coefficient Correlation and Coefficient Determination results for each metric

	Experiment: 8 Instances				Experiment: 40 Instances			
Metric	R	R^2	$Adj.R^2$	*p-value*	R	R^2	$Adj.R^2$	*p-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

Based on the explanation given the above, we chose $Adj.R^2$ as it was the most reliable measure to assess the relevance of each of monitoring metrics and the activities of the rolling upgrade operations. In other words, $Adj.R^2$ is chosen, as it is a less biased metric, to measure the prediction accuracy of a regression model, given the predictors (event logs) and target (monitoring metric) variable. Prediction abilities of the metrics based on the value of $Adj.R^2$ are shown in Fig. 5.9.

For several metrics, highlighted in bold in Table 5.2, we observe fairly strong values (> 0.5 for both experiments) 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, including CPUUtilizationAverage, CPUUtilizationMaximum, StartedInstances, and TerminatedInstances.

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. 5.9, 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 *TerminatedInstances*. In this case, $R^2 = 0.912$ and $Adj.R^2 = 0.912$ are statistically significant at confidence level $p < 0.05$. A similar interpretation of the model can be inferred from Table 5.2 for the experiments with eight instances, and 40 instances for other metrics.

The results show that seven metrics have a *p-value* greater than .05, indicating that our regression analysis does not show a good fit for these metrics. Therefore, they are not valid candidates for our anomaly detection. 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 ($Adj.R^2$) 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*.

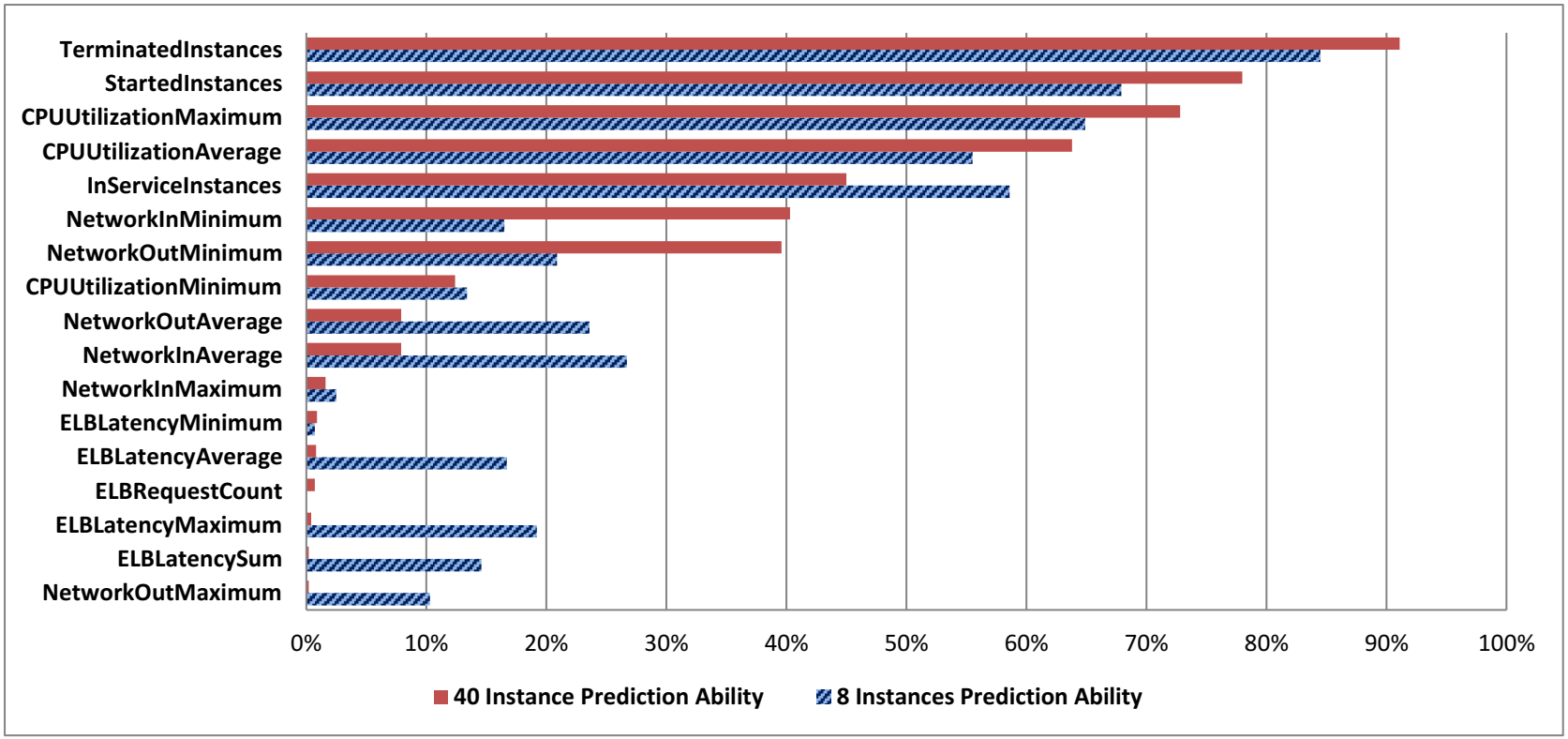


Figure 5.9: Prediction ability for each monitoring metric, based on $Adj.R^2$

5.4 Assertion Derivation for Anomaly Detection with Log-Metric Causality Learning

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 objective 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 more details, 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.

5.4.1 Causality Analysis and Assertion Derivation: Impact of Log Activities on Selected Metrics

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 5.3 we found the metrics that have correlation with the *operation's activities* overall, yet we need to find out which of the *operations' activities* are affecting a target metric, to derive assertion specifications for our anomaly detection. In other words, we know from correlation analysis that running rolling upgrade operation significantly affect few metrics such as the number of Terminated or Started Instance and CPU Utilization, yet we don't know which of the activities reported in Table 5.1 have a contribution to the changes we observe in the resource metrics.

Based on above, we thus seek to distinguish the activities of the cloud operation that are likely to affect one of the target metrics. In order to perform such analyses, we considered the correlation coefficient results for each predic-

Table 5.3: 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.

tive 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.

We performed regression analysis and Table 5.3 shows the coefficient correlation result for each operation's activity. By considering the *p-value*s in Table 5.3, we observe that the correlation of the metric and activities A3 and A4 are statistically insignificant ($p > .05$). Therefore, we conclude that the corresponding two variables cannot explain the variation of the target variable. Further, the standardized regression coefficient (B) 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 5.4.

Given the large difference between Activity A2 (0.836) and Activity A5 (0.15), presented in Table 5.4, it can be concluded that Activity A2 is the main activity that has the highest correlation to the termination of an instance. The respective log messages of the activity confirm that the model was effective in correctly identifying the activities that are related to the termination of VMs. These findings can then be used to define an assertion specification, as

Table 5.4: 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	0.836	0.000
A5_Instance is ready	0.149	0.023	0.111	0.000

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

explained in Section 3.4.2, with the following equation:

$$PredictedTerminatedInstances_i = 0.051 + 1.197 * A2_i + 0.149 * A5_i \quad (5.1)$$

We applied a similar approach for the regression analysis of other candidate metrics. Fig. 5.10 shows the importance or relative contribution of each predictor for each candidate metric, similar to the one that is explained for TerminatedInstances in Table 5.3, Table 5.4 and Equation 5.1.

5.4.2 Suitability of Metric for Anomaly Detection

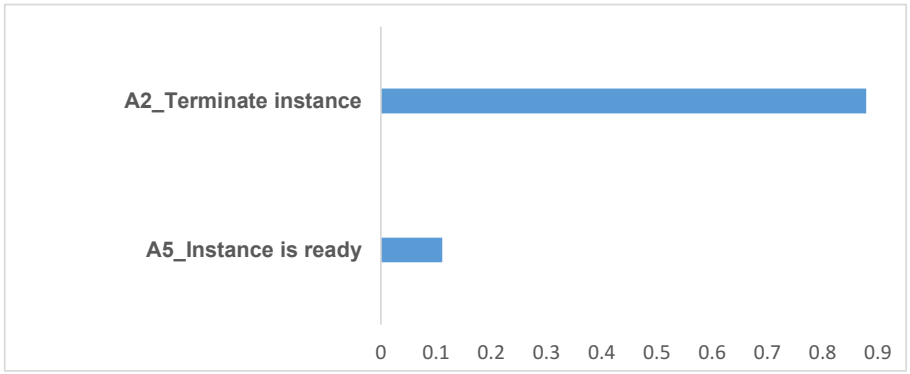
Our key anomaly detection objective articulated at the beginning of this section was: which of these metrics is most suitable to be leveraged for verification of successful execution of a rolling upgrade operation?

To find the answer to this question, two factors need to be considered:

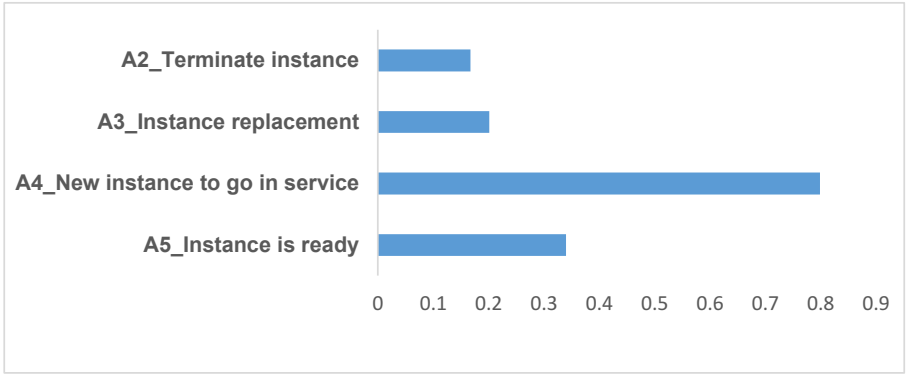
- (i) which metrics have the best prediction ability using statistical information?
- (ii) what types of metrics are best suited for the purpose of anomaly detection?

In regard 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 $Adj.R^2 = 0.911$ in Table 5.2.

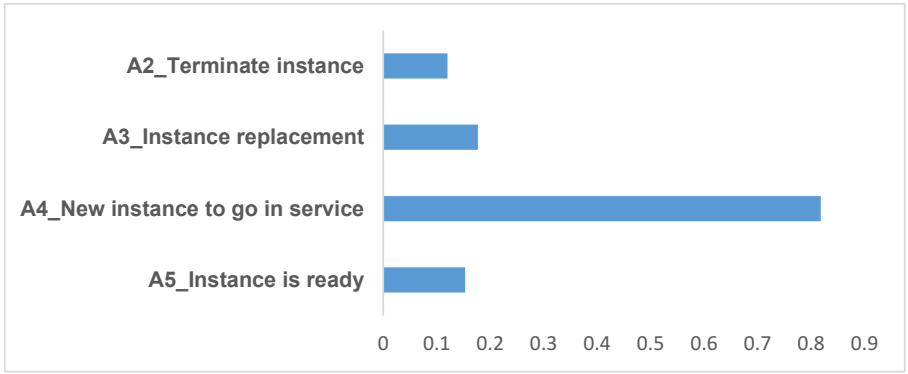
Such factual information can be a strong aid for an operator when filtering out all but the most relevant metrics, and to understand which metrics are affected by which activities in an operation.



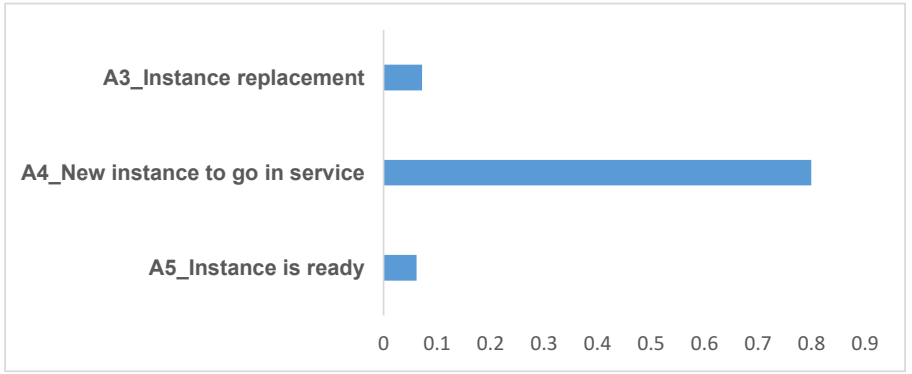
a) Predictors relative importance for TerminatedInstances



b) Predictors relative importance for StartedInstances

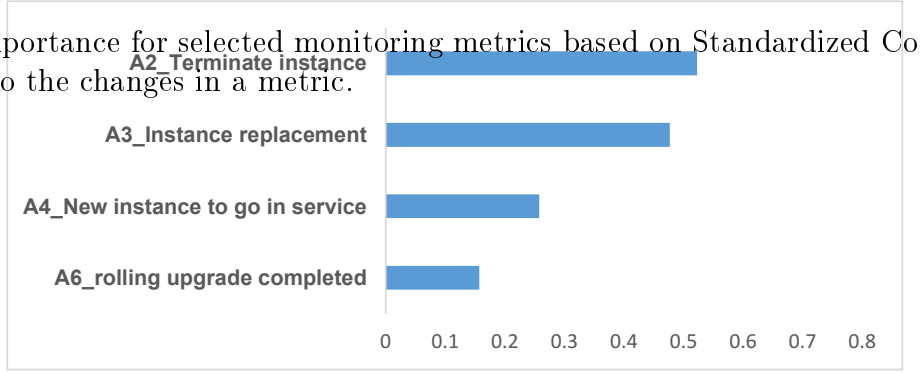


c) Predictors relative importance for CPUUtilizationMaximum



d) Predictors relative importance for CPUUtilizationAverage

Figure 5.10: Predictors' relative importance for selected monitoring metrics based on Standardized Coefficient(B) - larger value indicates higher contribution of an activity to the changes in a metric.



e) Predictors relative importance for InSeriveInstances

With regard to the second consideration, given the anomaly detection requirement for a rolling upgrade operation, state-based metrics that may reflect the target changes of the rolling upgrade operation should be the first candidates. This is important because our anomaly detection has the objective of finding unintended changes in the 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 candidates for anomaly detection. Nevertheless, for the sake of comparison between the results from state-based metrics (e.g. TerminatedInstances) and non-state-based metrics (e.g. CPUUtilization), we demonstrate the evaluation result of both types of metrics in the next section.

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

The reader may be concerned with 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 lead to very similar conclusions, although 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 long as we have sufficient data for a statistical analysis, we can be sure that even in the case of changing conditions, the result of identifying selected metrics and the log event predictors that affect a particular metric will be the same. For instance, the metric of TerminatedInstances will always show high correlation with the activities of rolling upgrade (Fig. 5.9), or A2 in logs (Fig. 5.10) will always have the highest impact on changing the TerminatedInstance metric compared to other activities. However, how much the impact

of activities on metrics will change if the system conditions or configuration changes needs further investigation.

Another question that may be asked is how precise the monitoring information must be in order for the approach to work. Inevitably, if the precision of the monitoring information is universally low, we might not be able to detect a statistically significant correlation between metrics and log events. However, first, this becomes clear during the training phase, and second, thanks to the emergence of the DevOps movement, the monitoring systems of most of the current private and public cloud solution providers have comparable operational monitoring tools to CloudWatch - a survey and comparison of cloud monitoring platforms is reported in [2].

As for the expressiveness of logs, the important point is that log events can be distinguished from one another, that is, it is possible to determine which particular event is represented by a log line. Besides timestamp information, we do not further use the actual content of log lines, and hence the required expressiveness is fairly low for our work.

5.5 Anomaly Detection Evaluation

In the previous sections, we investigated and discussed what steps should be taken to perform log analysis, map log events to metrics, select the most relevant metrics and derive a statistical model to learn the relationship between activities reported in logs and changes in resource metrics. The purpose of this section is to examine and evaluate the applicability of the proposed approach to detect anomalies.

Therefore, in this section, we describe the evaluation method and we explain how we applied our approach to error detection in the rolling upgrade case study. As argued in Chapter Two, our method is unsupervised – thus so far for selecting metric and deriving assertion specifications we train it only on data from *normal* instances of the operation processes, i.e. without injected faults. But in this section, as it has focused on the evaluation of the approach, 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 automatically distinguish them from direct effects of faults. Key insights and lessons learned from our experiments are discussed at the end of the section.

5.5.1 Evaluation Method

In order to evaluate how well the derived assertions (Section 5.4) can detect errors, we conducted a second experiment which was run independently from the one used to learn the model. In this direction, the experiments were conducted on Amazon EC2, upgrading eight instances, two instances at a 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 (Section 5.1.2).

As explained in Section 5.4, the equations derived from the multiple regression model in the learning phase can be used to predict the values for the target metrics, such as 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 anomaly is detected. In this direction, we wanted to find out how accurately the proposed approach in this thesis could identify anomalies.

We hypothesise that in any given time window, in case there are anomalies happening at runtime, our approach could identify most of these anomalies. We expect these anomalies, such as sudden termination of a VM or a peak on CPU usage, to be distinct from the impact of a rolling upgrade operation 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.

To this end, a total of 115 faults were injected in VMs involved in 22 rounds of rolling upgrade operation. Since the injected faults were all VM failures, our approach tries to distinguish between VMs being terminated due

Table 5.5: Classification metric for the generated alarm

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

to legitimate operational activity, and termination caused by faults. 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 here – the state of VMs is a prime source of anomalies, as discussed in Section 2.2. 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 six minutes by a software service that was running in parallel with the rolling upgrade operation. In our experiments, each fault was injected into the respective VM separately. There were cases where two VMs went out of service due to two separate fault injections within the same 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. Hence it is possible that more than one failure could occur during one rolling upgrade operation, thus, having concurrent faults injected in a rolling upgrade operation is more realistic than the opposite. When an anomaly is detected during a time window, an alarm is issued, containing the information about the difference between the expected value calculated from the regression equation based on log events versus the actual number of terminations that occurred in that minute, as indicated in the respective 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 5.5 explains these four categories in terms of an alarm being raised (or not), and a fault is 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 perfectly.

These four categories are the basis for calculating precision, recall, and the F-measure [83]. Precision is a measure to assess the exactness of the result, i.e. the percentage of the valid issued alarms out of all issued alarms.

$$Precision(P) = \frac{TP}{TP + FP}$$

Recall is a measure of completeness of correct alarms, i.e. the percentage of injected faults where an alarm was raised.

$$Recall(R) = \frac{TP}{TP + FN}$$

F-measure (or F_1-score) is the weighted average (harmonic mean) of precision and recall.

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

5.5.2 Evaluation Result with State-Based Metric

Given the discussion in Section 5.1.4 regarding direct and derived metrics, in this section, we will focus on evaluating the proposed approach on a state-based derived metric.

In order to evaluate the anomaly detection, we employ the assertion equation derived from Equation 5.1 and run the anomaly detection for all the available data records. As our metric collection had one-minute time intervals, the prediction also calculated at each minute and compared with the actual value of the metric. We labelled the results of this default time window as zero minute time window (0mTW). The precision, recall and F-score of running 22 rounds of rolling upgrade operation with 115 injected faults of 0mTW are shown in Table 5.6.

In our study, we observed possible delays between an operation action and its effect(s) becoming observable which may not be reflected in 0mTW. 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 three minutes. It is thus not uncommon that a VM is terminated in one minute, but the CloudWatch metrics reflect only the

Table 5.6: Evaluation results of state-based metric (TerminatedInstances) – 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

termination in the next minute, or possibly later. This delay is observable in the actions of legitimate operations, 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, 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. Please note 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 F_1-score increases by x?

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 5.6. Precision, recall, and F_1-scores are given without considering the impact of the ripple effect of injected faults. It may be noted 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 comparatively smaller. The recall changes in a similar fashion.

These observations can be explained because of the time delay that the action of termination takes to be completed: the majority of terminations are completed either within the current minute or the next minute – it rarely takes longer 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. Ripple effects may occur

Table 5.7: 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.
2	Instance is terminated by fault injection while waiting to be started.

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. Table 5.7 shows three types of *ripple effects* we observed in the experiment, as well as their numbers of occurrence.

The first two types of ripple effects 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 5.6).

To distinguish actual failed predictions from ripple effects (where the prediction behaved as expected), we analyzed all 30 FP and seven FN 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 apply

Table 5.8: Evaluation results with state-based metric (TerminatedInstances) – detection result with manual ripple effect re-classification

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

a ripple effect detection algorithm to classify them as TP/TN, leading to the results shown in Table 5.8 and in Fig. 5.11 respectively. In Section 5.5.4 we discuss a method for automatic ripple effect detection.

5.5.3 Evaluation Result with Non-State-Based Metric

In the previous section, we demonstrated and discussed anomaly detection performance utilizing TerminatedInstance as the main monitoring metric. In our metric selection process (cf. Section 5.3), we observed that the non-state-based (direct) metrics of CPUUtilizationMaximum and CPUUtilizationAverage also have a fairly good correlation with the rolling upgrade log activities, though not as high a correlation as 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 these CPU-related metrics, and compare this with the results we obtained when using the TerminatedInstances metric.

In the approach for the state-based metric, the threshold value is an indicator of whether a change of state occurred or not, and whether that matches the normal behaviour of the system. For non-state-based metrics such as CPU utilization, a threshold indicates the value that separates outliers from the range of normal values of the metric. In most statistical-based anomaly detection techniques, standard deviations (σ) from the mean are used to detect outliers. Often the values dispersed above $\pm 2.5\sigma$ to $\pm 3.0\sigma$ are considered outliers [102]. 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 CPU utilization. Any observed values above this range are considered

anomalies. In our experiment, we consider the differences of the predicted and the actual value of CPU utilization that are within accepted standard error of estimate and smaller than one standard deviation $\pm 1\sigma$ from mean to be normal, otherwise, an alarm is registered. The use of tighter thresholds than in the literature becomes possible in our setting, since we gain additional precision from analyzing the process context.

As in the experiments for TerminatedInstances, we have used the two separate datasets for learning and evaluation. Both the learning and the evaluation system were monitored and exposed to a workload averaging approx. 40% CPU utilization of the respective EC2 Auto Scaling Group's aggregated CPU power, with a standard deviation of 2.05% CPU utilization. The systems under test for evaluation were also exposed to an additional CPU workload task of 20% CPU utilization on average. This additional load was injected periodically: it lasted between two and three minutes, followed by no additional load for two minutes, and then repeated.

In contrast to regular anomaly detection methods in the literature, where the metrics are the main source of information, we have contextual information from the operation, and we obtained the approximate effects of operation activities on resource consumption from the learning dataset through 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 (cf. Fig. 5.10) while A2, A3 and A5 had very low impact on CPU Utilization, and A1 and A6 did not show any observable impact. Our approach accounted for the impact of these activities, based on Equation 5.1, by means of which we learned the assertion from the learning dataset. With this assertion, we predicted values for the evaluation dataset and raised alarms where the prediction did not match the observed metric values. Then we classified each time window as TP, TN, FP, and FN as before, and calculated precision, recall, and the F_1-score. The results are given in Table 5.9, 5.10 and Fig. 5.11

The precision and recall for a zero minute time window, as shown in Table 5.9, 5.10 and Fig. 5.11, are very low for both CPUUtilizationAverage and CPUUtilizationMaximum. When expanding the time window to one minute

Table 5.9: Evaluation Result with Non-State-Based Metrics for CPUUtilizationMaximum

	CPUUtilizationMaximum				
Metric	0mTW	1mTW	2mTW	1mTWRipEff	2mTWRipEff
Precision	0.212	0.399	0.595	0.427	0.614
Recall	0.728	0.798	0.917	0.814	0.918
F-Score	0.329	0.532	0.721	0.560	0.736

Table 5.10: Evaluation Result with Non-State-Based Metrics for CPUUtilizationAverage

	CPUUtilizationAverage				
Metric	0mTW	1mTW	2mTW	1mTWRipEff	2mTWRipEff
Precision	0.235	0.433	0.701	0.464	0.716
Recall	0.833	0.789	0.895	0.828	0.914
F-Score	0.367	0.559	0.786	0.594	0.803

before and after (1mTW), the results improve. But the best results were obtained by using two-minute time windows (2mTW) as most of the detection happened within one minute time window, it was expected expanding the time window further to two minutes improve the result slightly more. Based on our analysis and understanding, we hypothesize that there are two reasons for this observed delay of the effect of VM failures based on CPU metrics. The first reason is similar to an effect on TerminatedInstances: a VM that is in state "terminating" may still send metrics to CloudWatch for some time within the tens of seconds.

The second hypothesized reason is the cycle that an instance going into service experiences: (i) While it boots up, its CPU utilization is 100%; at some point during boot-up, CloudWatch data starts being collected for this instance, and registered for the Auto-Scaling Group. (ii) Once boot-up is completed, the CPU utilization drops to less than 10%, as the VM does not yet receive requests. (iii) Finally, the machine is registered with the load balancer and receives requests. Soon after that, its CPU utilization becomes similar to that of the other machines in the ASG. Due to (i) and (ii), the start of a VM can have a strong distorting effect on the CPU utilization, in particular, CPUUtilizationMaximum can be distorted by (i). The whole procedure typi-

cally completes within less than two minutes for the VM configuration we used, and hence we hypothesize that the effect becomes less impactful for detection with 2mTW.

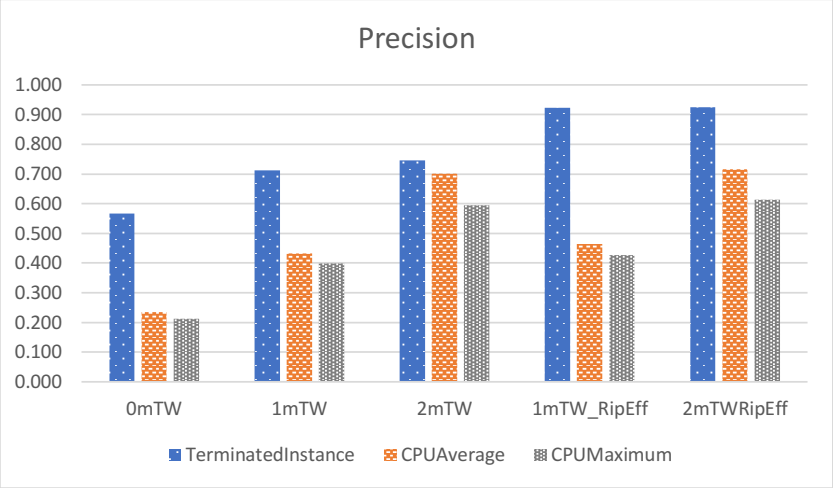
It may be noted that there is a considerable gap between anomaly detection delay between TerminatedInstance and CPUUtilization. This observation can be explained as follows. For the case of detecting anomalies with TerminatedInstance, we had the information to decide when the termination happened and whether it was the result of a legitimate process or our fault injection, and that helped us to detect failures as soon they occurred. In contrast, when we attempt to detect a failure from metrics based on CPU utilization, there are additional factors that complicate detection further.

5.5.4 Ripple Effect Detection

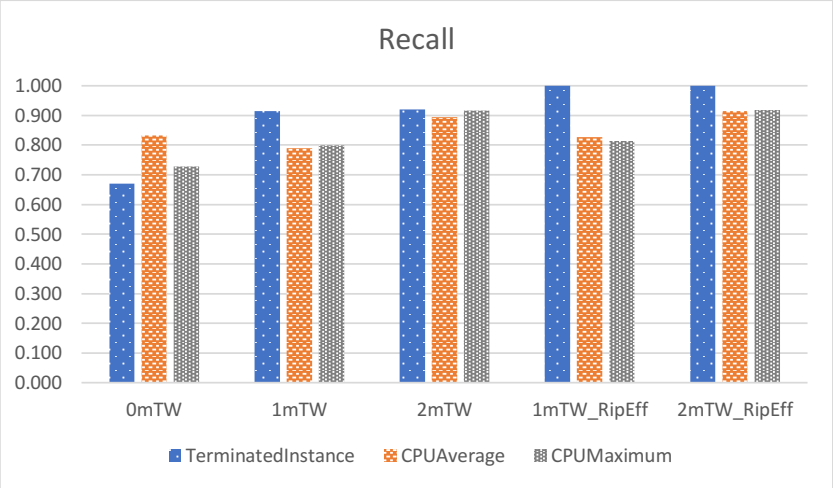
In the previous subsections, we discussed the presence and sources of ripple effects of errors, and how we manually identified them as anomalies. We showed that our approach achieves high accuracy of detecting injected failures when ripple effects are accounted for.

Anomalies detected in a system are reported through 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. The remaining open question therefore is: how can we detect ripple effects automatically? In more detail, how can we distinguish if a detected anomaly stems directly from an error vs. from the ripple effect of an error? If we can answer this question, we can suppress alerts from the latter.

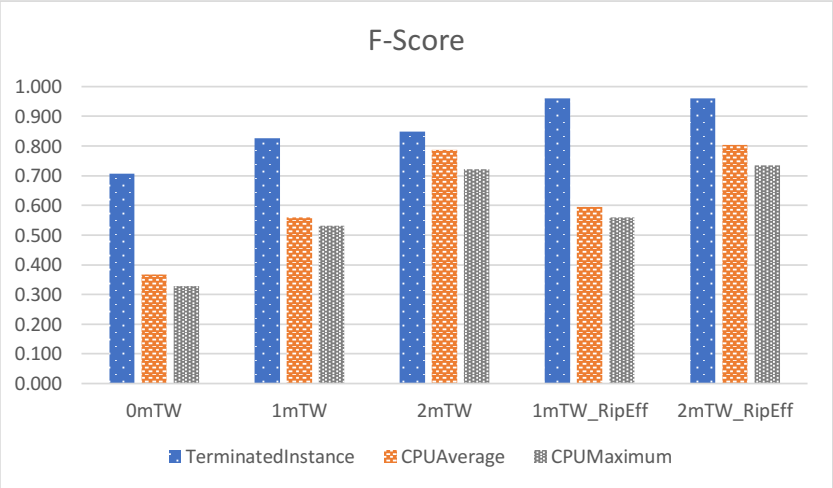
In this section, we describe our approach to addressing this issue, i.e. a mechanism to automatically detect ripple effects of errors. To this end, we kept track of the instance identifiers, which are present in both the metrics data and in the operation's logs. Additionally, we already had the timestamps of the event logs – e.g. the time that the termination of an instance had been triggered. Given this information, as well as the records of detected anomalies, we determined if a raised anomaly is related to an already affected VM or not;



a) Precision



b) Recall



c) F-Score

Figure 5.11: Results for three different time windows and with ripple effects for CPUAverage and CPUMaximum vs. TerminatedInstance.

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
```

if not, we tag the anomaly as a ripple effect. The pseudo-code of this ripple effect detection algorithm is shown as Algorithm 1. In the implementation of this method, if our method decides that a detected anomaly is a direct failure, we raise an *error alert*; if it is a ripple effect, we raise a *warning alert*. Suppressing the latter is a simple configuration change.

We then applied the implementation of Algorithm 1 to the entire fault injection experiment. First, we re-ran the detection with automatic ripple effect detection enabled for fault detection, based on the metric TerminatedInstances.

The three types of ripple effects we observed in our experiments are shown in Table 5.7. 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. Their presence is thus an effect of the experimental set-up and should not be taken into account. Therefore, we did not count these at all.

Out of the remaining 23 ripple effects, our automatic ripple effect detection 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, and they bypassed our automatic ripple effect detection.

For the CPU utilization-based detection, the technique applies in the same way: any of the 21 FPs caused by a ripple effect that we could detect was corrected. If a ripple effect was not detected in the first place, it was already classified as a TN – in other words, the fact that the error detection missed it actually played out in its favour, i.e. higher precision. For detection based on CPUUtilizationMaximum with 2mTW, this was the case in 16 cases. For 2mTW and CPUUtilizationAverage, this case occurred 18 times. The automatic ripple effect detection here worked correctly in the remaining five and three cases, respectively.

5.6 Summary and Lessons Learned

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 operations' 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 of 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 integrating the 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 operational processes.

Chapter 6

Flight Data Processor Case Study

In the previous chapter, we evaluated different stages of the proposed approach of this thesis in a comprehensive case study of rolling upgrade with different configuration settings. In this chapter, we extend our evaluation in order to assess the applicability of the proposed approach for second case study on Flight Data Processor(FDP).

We will follow the same steps in our framework that were explained in Chapters 3 and 5, thus, we give a concise explanation of each step while providing more explanation about the nature of the case study, the differences between this case study and the rolling upgrade case study, and the results obtained from running the experiments.

Similar to the previous chapter, we aim to evaluate to what extent we can answer the research questions of this thesis. In particular, we are looking to investigate the applicability and generalisability of our approach in a different environment from the previous case study of rolling upgrade. The case study used in this chapter has key differences with the case study of rolling upgrade, including environment, the scale of monitoring data and the format and structure of logs and metrics, which some of these differences are explained below.

- The previous case study was based on a public cloud provider (Amazon EC2), while in this case study, the service is hosted on private in-house servers.

- In contrast to having the well-developed and systematic monitoring solution of CloudWatch, monitoring in this case study relies on native Linux operating system monitoring mechanism. In particular, the granularity of the sampling rate of resource monitoring for these two settings are significantly different: a one-minute sampling rate in CloudWatch versus a 100-millisecond sampling rate for this case study.

- Rolling upgrade is a sporadic operation which is often executed with irregular frequency and is an administrative operation, while this case study focuses on a middleware operation which is a type of ongoing operation used for an application service.

- Most importantly, in terms of the scale of log file, the middleware log used in this case study is significantly larger compared to the rolling upgrade case study. In particular, this case study has over eight times more unique log event types.

6.1 Experimental Set-Up

The data of this project and the description given in this section has been made available to the author of this thesis by Raffaele Della Corte, Marcello Cinque, and Antonio Pecchia from Dipartimento di Informatica e Sistemistica - Università degli Studi di Napoli Federico II. The content of this section originates from [26] and personal communication.

System Overview

The system used for this case study is a middleware platform for the integration of mission-critical distributed systems for an air traffic control (ATC) system. The middleware is a complex modular Java system, which allows integrating a variety of legacy ATC applications, such as flight data processors (FDPs) and controller working positions (CWPs). The middleware runs on the JBoss[1] application server.

[1] http://jbossas.jboss.org/

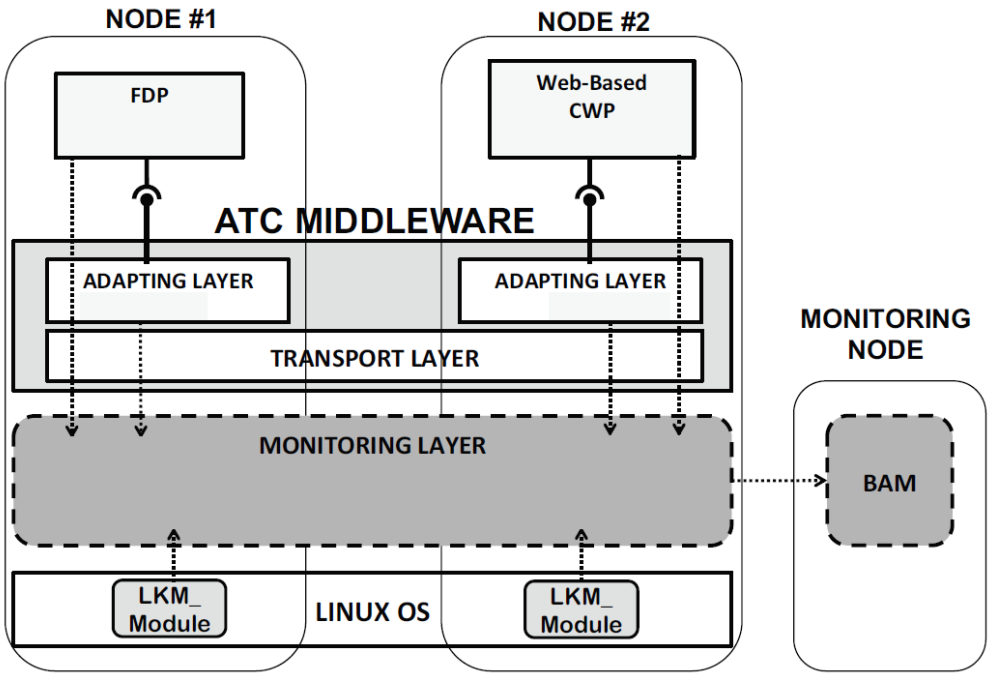


Figure 6.1: High-level architecture of the middleware prototype instrumented with the monitoring system

The high-level architecture of the prototype is shown in Fig. 6.1, together with the deployment of the monitoring service. The prototype has been provided with two ATC applications, that is, a flight data processor application, which generates and updates flight data (i.e., data that describe a flight, such as arrival and departure time and flight trajectory) and publishes the data on the middleware, and a web-based controller working position, which receives the flight data from the middleware and presents it on a web console.

The middleware platform consists of transport and adapting layers. The transport layer ensures the communication between the FDP and CWP applications, according to the publish-subscribe paradigm. The adapting layers allow applications FDP and CWP to use the middleware and its services. The Business Activity Monitor (BAM) is a service that automates the collection of system probes and execution data, allowing it to monitor the behaviour of systems consisting of several nodes/services running on Linux operating systems.

The data collection from system probes is conducted by the monitoring agent using the loadable kernel module (LKM) of the Linux operating system. The Linux kernel stores all the information about each running process into

a process descriptor, which contains information such as data describing the open files of the process, its state and its resources usage (e.g., CPU and RAM).

The middleware has been deployed in a configuration consisting of two nodes (two separate virtual machines of Intel Xeon E5-1620 v2 - 8 cores, 16 GB RAM, 1 GB/s network interface and running Ubuntu 14.04) that emulate an ATC system. The testbed is shown in Fig 6.1. Two nodes run (i) an FDP application and a web-based CWP application, respectively, (ii) an instance of JBoss application server with the middleware, and (iii) the monitoring service which hosted on a separate machine, allowing monitoring data collection and visualisation.

It is worth noting that while the experimentation has been conducted in a controlled testing environment, the applications emulate a real ATC system. The source code of the FDP and adapting layers of the middleware have been instrumented by means of rule-based logging; the OS processes running the FDP and the middleware are monitored by the kernel probes collected with a period of 100 ms.

The FDP both generates the data and updates the flight information. The web-based CWP receives data from the middleware and presents flight information on a web console. The system was exposed to simulated workload over a duration of around six minutes, producing around 4,000 log lines.

To force the collection of data under failure conditions, errors were emulated into randomly selected pieces of code of the FDP application. As with experimental organisation, first, monitoring data were collected in a normal (error-free) mode, then three types of errors were emulated, including active hangs that are emulated by triggering an infinite loop, passive hangs that are emulated introducing a wait on a locked semaphore, and crashes which are induced by deliberately dereferencing a null pointer. In this process, for each experiment, the middleware and the applications are started. Then the systems are exposed to a normal workload. Next, error emulations are performed 2-3 minutes after the beginning of the run.

The focus in our analysis in this chapter is on JBoss middleware logs of the FDP machine, and available resource monitoring data. The normal run of the

operation is used as the source for the learning phase, and the operation runs that contain errors for anomaly detection evaluation.

6.2 Log Analysis

This section describes how we process log files, identify unique log event types, represent log events in a quantitative form and cluster highly correlated log event types to a set of log activities.

The prerequisite step to process logs in our approach is to make sure that log events include timestamps. Timestamps are necessary to track log events and also to map the occurrence of log events to metric observations. Unlike the rolling upgrade logs, where all log events were presented in separate log lines and they had timestamps, in this case study we found that some log event types are dispersed across multiple lines, as shown in Listing 6.1. To prepare the log for further analysis, we implemented a module to standardize the log events, in which all of them contain a timestamp and each log event is encapsulated in only one log line.

Listing 6.1: Example of log event reported in multiple lines

```
Line1: 2015-09-28 12:24:44,332 INFO  [SWIM_SUIT.SDS] (http-swim
   -host2F192.168.0.52-8180-1) [SWIMSharedDatastoreBean]
   getData() : trying to retrieve data with key:
Line2: dataDomain: FDD
Line3: dataIdentifier : :
Line4:  stakeholder_ID: SELEX_TWR_LIMC
Line6: dataType: com.selex.swim.fdd.sharedData.
   FDDSharedDataKind@3f54dda7
```

6.2.1 Log Event Type Extraction

To observe how the activities reported in log events change the state of resources, we are interested in tracking the occurrence of log events. We needed a way to parse and trace log events. As we explained in Chapter 3, we employ regular expressions to derive a template of event logs. At this stage, the goal of log parsing is to extract the pattern of recurring event logs by automatically

separating the constant and variable parts of a raw log message, and further transform each log message into a specific log event type. In this process of regular expression extraction, the raw log messages are transformed into a set of unique structured log events. For each new log line, the log event is compared by pattern matching with regular expressions, and if a pattern is not found, the above steps are repeated for the new log line.

The middleware server log files in our case study have around 4,000 log lines. Extracting regular expressions and finding unique event types in such a volume of log lines cannot be done in a manual way. Therefore, we employed a tool called POD-Discovery to generate regular expressions for unique log events. POD-Discovery is a log abstraction tool [130] which can be used to cluster low-level event traces into higher-level events. The desired level of abstraction from low-level logs to higher-level logs depends on the system to be monitored.

Similar to our approach for the rolling upgrade case study, POD-Discovery, as its core concept, tokenises each log message to several tokens. The tokens of a log message can be divided into two parts: the constant parts and the variable parts [54]. The constant tokens of a recurring log event remain the same for a recurring log event by default, while the variable tokens hold the runtime information of a recurring log event, like an IP address or a port number.

In the process of regular expression extraction, the constant tokens are used as an exact string, while the variable tokens are expressed with a regular expression that matches the variable parts. For example, the log message containing "Class Loader Updated" will be expressed as "Class\sLoader\sUpdated", where "\s" matches whitespace (spaces, tabs and new lines). As an example of a variable token, "Thread-37" will be defined as "Thread-\d+" where "\d+" indicates that the thread ID can be of a number with one or more digits. POD-Discovery provides a configuration file where the variable tokens can be defined with regular expressions, and by means of this, all the variable tokens are replaced with the allocated regular expression in the process of log transformation. We used this feature to customize POD-Discovery for the this case study.

POD-Discovery employs a token distance measure using Levenshtein distance [77] for string comparison. This method is used as a metric to know how many similarities exist between a token of one log event and another one. This tool provides a scale for adjusting the desirable similarity distance measure, which can be used as a threshold to cluster low-level logs to a higher-level abstraction. Fig 6.2 shows a snapshot of the dendrogram diagram generated from processing FDP middleware server logs. Yellow circles show the tree nodes at 10% similarities, and selecting each node shows the log events under the tree hierarchy of that node at the bottom of the screen.

For pattern extraction of unique log events from logs in our study, we began to set a distance measure of minimum similarities and we generated log clustering. Our initial inspection of clustered logs at the minimum level showed that they were too low, as there were similar log events dispersed over two or three activities. After trying this and a few rounds of inspection and gradually increasing similarity levels, we found the 20% distance threshold gives us a level which grouped highly similar log events into individual groups. Fig 6.3 shows a snapshot of the log clustering output. Each cluster/section contains similar log events that refer to one log event type. Once a desirable log similarity threshold is chosen, the clustered logs can also be inspected manually and wherever necessary they can be fine-tuned by either combining two or several clusters together or by splitting the logs of one cluster into smaller ones. Fig 6.3 shows a snapshot of the log events of FDP logs that form unique log event types. Further details about the POD-Discovery tool can be found in [130].

In order to extract a regular expression pattern that identifies unique log event types, we processed the sanitised (pre-processed) middleware server log file with POD-Discovery at 20% similarity distance. The output of processing 4,000 lines of middleware server log led to 155 unique log event types. The above outcome, especially compared to the rolling upgrade case study where we were dealing with 18 event types, demonstrates that employing the above-described method significantly facilitates the processing of a large log file for the extraction of a fairly large number of log event types.

Figure 6.2: A screenshot of POD-Discovery - yellow circles show the tree nodes at 10% similarities, the selected node shows the log events under the tree hierarchy of that node at the bottom of the screen

```
2910-09-28 12:29:49,027 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDUtilityServiceImplBean][unsubscribe] Pre-Entity : LH--
2935-09-28 12:29:49,352 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDUtilityServiceImplBean][unsubscribe] Pre-Entity : LH--

 251-09-28 12:25:00,671 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDBusinessServicebean][createFO] FlightObject Identifier calculated : LH--
1009-09-28 12:26:16,851 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDBusinessServicebean][createFO] FlightObject Identifier calculated : LH--

 250-09-28 12:25:00,671 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDBusinessServiceBean][getFlightIdentifier] Flight identifier build: LH--
1008-09-28 12:26:16,851 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDBusinessServiceBean][getFlightIdentifier] Flight identifier build: LH--
1158-09-28 12:26:17,441 [SWIM_SUIT.FDD] (http-swim-host%F...--) [FDDBusinessServiceBean][getFlightIdentifier] Flight identifier build: LH--

 730-09-28 12:25:31,478 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
 997-09-28 12:26:01,756 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
1581-09-28 12:26:47,333 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
1844-09-28 12:27:17,703 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
2101-09-28 12:28:03,031 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
2347-09-28 12:28:33,359 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
2626-09-28 12:29:03,687 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat
2891-09-28 12:29:33,986 -- ::, WARN [SWIM.FDDConnector] (WorkerThread#[...:]) [FOManagerImpl][addFlightObject] ATTENTION: provided FOService LH-- was already saved --> updat

2965-09-28 12:29:49,394 -- ::, SEVERE [MESSAGING.PubSubService] (http-swim-host%F...--) [DDSPublisher][closePublisher] Publisher has been removed for topic type FOSummary
2977-09-28 12:29:49,399 -- ::, SEVERE [MESSAGING.PubSubService] (http-swim-host%F...--) [DDSPublisher][closePublisher] Publisher has been removed for topic type FOSummary

1815-09-28 12:23:58,699 -- ::, WARN [org.jboss.wsf.stack.cxf.deployment.aspect.DescriptorDeploymentAspect] (HDScanner) Spring not available, skipping check for user provided

1115-09-28 12:23:59,228 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
1155-09-28 12:23:59,247 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
1285-09-28 12:23:59,537 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
1305-09-28 12:23:59,543 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
1345-09-28 12:23:59,546 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
1385-09-28 12:23:59,576 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
1415-09-28 12:23:59,634 -- ::, WARN [org.jboss.ejb.TimerServiceContainer] (HDScanner) EJBTHREE-: using deprecated TimerServiceFactory for restoring timers
```

Figure 6.3: A snapshot of POD-Discovery output for identifying log event types

6.2.2 Representing Log Event Type as Quantitative Metric

In the previous section, we extracted the regular expressions that represent unique event types. In Chapter 5 for the rolling upgrade case study, the monitoring data from Amazon Cloudwatch was available at not less than one-minute intervals. Therefore, we used a one-minute time window for mapping the activities of logs, based on their timestamps.

Middleware logs in the case study of this chapter are available with the precision of milliseconds, however, the occurrence of logs varies between a few milliseconds to a few seconds. Monitoring metrics are reported in near[2] 100-millisecond intervals. Given the interval occurrence of logs and the availability of the data, we decided to choose a one-second interval as the default time window. This time window is small enough to provide fine-grain monitoring and not so small that the lasting impact of the operation's action on resources would lead to too many false alarms.

Similar to what we explained in Chapter 3, Section 3.3.1 and for Chapter 5, Section 5.2.1, we extract a metric that shows the occurrence of log events based on the interpolated occurrence strength of each event type. The difference between the rolling upgrade case study and this case study is the size of the time window.

As discussed in Chapter 3, given a log event type is denoted as e, the smallest unit of time that logs can track is denoted as x, the interval of time that monitoring metrics are available is denoted as tw, and D_{tw} represents the duration of time window, then the weight-timing occurrence of an event type at time x of a current time window and the next time window can be obtained:

$$e_{n(tw)} = \frac{D_{tw} - x}{D_{tw}} \qquad e_{n(tw+1)} = \frac{x}{D_{tw}}$$

So for the sum of n times occurrences of an event type for a time window, we have:

[2]Actual interval varies between 100 and 105 milliseconds.

$$E_{tw} = \sum_{i=1}^{i=n} e_{n(tw)}$$

The process of deriving log counting metrics based on the above is automatically executed using our anomaly detection prototype. For instance, given these two log events:

```
2015-09-28 12:26:16,562 INFO  [SWIM_SUIT.FDD] (Thread-37) [
   FDD_FlightDataListener_Impl] Class Loader Updated

2015-09-28 12:26:16,974 INFO  [SWIM_SUIT.FDD] (Thread-37) [
   FDD_FlightDataListener_Impl] Class Loader Updated
```

the two log events from the same event type here occurred two times within the one-second time window in the timestamp *12:26:16* at 562 and 974 milliseconds, respectively. We obtain the interpolated occurrence strength as below:

$$e1_{12:26:16} = \frac{1000 - 562}{1000} = 0.438 \qquad e1_{12:26:17} = \frac{562}{1000} = 0.562$$

$$e1_{12:26:16} = \frac{1000 - 974}{1000} = 0.026 \qquad e1_{12:26:17} = \frac{974}{1000} = 0.974$$

Then the interpolated occurrence strength for e1 in the current time window and the next time window are obtained as follows:

$$E1_{12:26:16} = 0.438 + 0.026 = 0.464$$

$$E1_{12:26:17} = 0.562 + 0.974 = 1.536$$

As the result shows, the impact of the occurrence of log events is distributed between two time windows (two seconds), where we have a higher interpolated occurrence strength for the second time window. Similar steps are applied automatically using a Java module, and thereby the metrics of the interpolated occurrence strength of log event types are derived for all the 155 event types

at each second. Fig 6.4 shows a snapshot of few records of this matrix: E27 to E48 show event types 27 to event type 48, and the zero value indicates no occurrence of the event type within that timestamp.

Timestamp_3	E027	E028	E029	E030	E031	E032	E033	E034	E035	E036	E037	E038	E039	E040	E041	E042	E043	E044	E045	E046	E047	E048
2015-09-28_12:23:58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:23:59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:03	0	0	0	0	0	0	0.52	0.26	1.98	0	0	0	0	0	1.79	0	0	0	0	0	0	0
2015-09-28_12:24:04	0	0	0	0	0	0	0.43	1.13	0.89	0	0.45	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:05	0	0	0	0	0	0	0	0.9	0.79	0	0.65	0	0	0	0	0	0	0	0	2.97	0	2.96
2015-09-28_12:24:06	0	0	0.58	0	0	0	0	0	0	0	0.98	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:08	0.41	0	0	0	0.58	0	0	0	0	0	0	0	1.33	0	0.99	0	0	0	0	0	0	0
2015-09-28_12:24:09	0	0	0	0	0	0.45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:10	0	0	0	0	0	0.18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:11	0.72	0	0	0	1.43	0	0	0	0	0	0	0	2.14	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:12	0	0	0	0	0.74	0	0	0	0	0	0	0	0.92	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2015-09-28_12:24:15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6.4: A snapshot of the log event type matrix of interpolated occurrence strength of each event type at different timestamps

6.2.3 Log Event Type Correlation Clustering

In the approach of this thesis, we are interested in finding the log event types where their occurrences are highly correlated, regardless of the similarity or the dissimilarity of their log message. In the previous sections, we derived a metric of interpolated occurrence strength of log events. In this section, we use these metrics to find the correlation between log events. In this direction and similar to the rolling upgrade case study, we leveraged the Pearson correlation coefficient to identify the highly correlated log event types.

We defined a rule for event types to be grouped together when they had a correlation strength of more than 75% (Pearson-r > 0.75), where the values are shown to be statistically significant (i.e. $p\text{-}value < 0.01$). 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. Fig. 6.5 shows a portion of the correlation matrix generated from the interpolated occurrence strength of log event types.

As a result of our correlation analysis, the event types are grouped into 17 log activities. The number of log event types associated with a group varies widely from one event type in a group to over 20 event types in a group. We named these event type groupings from Log Activity 01 until Log Activity 17.

It is worth noting that in this process of log analysis and clustering, we relied on statistical analysis only, so we neither analysed the context of the log nor used domain knowledge for clustering log events.

		E001	E002	E003	E004	E005	E006	E007	E008	E009	E010	E011	E012	E013	E014	E015	E016	E017	E018	E019	E020	E021	E022	E023	E024	E025	E026	E027	E028	E029	E030
E001	Pearson Correlation	1	1.000	-.005	-.002	-.006	-.002	-.006	-.003	-.003	.726	-.008	-.007	-.007	.605	.558	-.006	.201	-.002	.441	-.002	.457	-.007	-.009	-.003	.586	-.002	-.002	.163	-.002	-.002
	Sig. (2-tailed)		0.000	.913	.964	.897	.964	.891	.949	.951	.000	.862	.887	.885	.000	.000	.897	.000	.964	.000	.964	.000	.885	.850	.949	.000	.964	.964	.000	.964	.964
E002	Pearson Correlation	1.000	1	-.005	-.002	-.006	-.002	-.006	-.003	-.003	.726	-.008	-.007	-.007	.605	.558	-.006	.201	-.002	.441	-.002	.457	-.007	-.009	-.003	.586	-.002	-.002	.163	-.002	-.002
	Sig. (2-tailed)	0.000		.913	.964	.897	.964	.891	.949	.951	.000	.862	.887	.885	.000	.000	.897	.000	.964	.000	.964	.000	.885	.850	.949	.000	.964	.964	.000	.964	.964
E003	Pearson Correlation	-.005	-.005	1	-.005	-.014	-.005	-.015	-.007	.191	-.007	.105	-.016	-.016	.036	-.015	-.014	.623	-.005	.152	-.005	.191	-.016	-.021	-.007	.114	-.005	-.005	.000	-.005	.217
	Sig. (2-tailed)	.913	.913		.913	.755	.913	.740	.877	.000	.877	.021	.731	.726	.429	.736	.755	.000	.913	.001	.913	.000	.726	.647	.877	.012	.913	.913	.991	.913	.000
E004	Pearson Correlation	-.002	-.002	-.005	1	-.006	-.002	-.006	.703	-.003	-.003	-.008	-.007	-.007	-.005	-.006	-.006	-.007	-.002	-.008	-.002	-.003	-.007	-.009	-.003	-.004	-.002	-.002	-.008	-.002	-.002
	Sig. (2-tailed)	.964	.964	.913		.897	.964	.891	.000	.951	.949	.862	.887	.885	.910	.889	.897	.881	.964	.854	.964	.951	.885	.850	.949	.937	.964	.964	.867	.964	.964
E005	Pearson Correlation	-.006	-.006	-.014	-.006	1	-.006	-.018	-.008	-.008	-.008	-.023	-.019	-.019	-.015	-.018	-.017	-.019	-.006	-.024	-.006	-.008	-.019	-.025	-.008	-.010	-.006	-.006	-.022	-.006	-.006
	Sig. (2-tailed)	.897	.897	.755	.897		.897	.695	.855	.862	.855	.620	.684	.679	.748	.691	.712	.671	.897	.601	.897	.861	.679	.589	.855	.822	.897	.897	.634	.897	.897
E006	Pearson Correlation	-.002	-.002	-.005	-.002	-.006	1	-.006	.710	.453	-.003	-.008	-.007	-.007	-.005	-.006	-.006	-.007	1.000	-.008	1.000	-.003	-.007	-.009	-.003	-.004	-.002	1.000	-.008	1.000	-.002
	Sig. (2-tailed)	.964	.964	.913	.964	.897		.891	.000	.000	.949	.862	.887	.885	.910	.889	.897	.881	0.000	.854	0.000	.951	.885	.850	.949	.937	.964	0.000	.867	0.000	.964
E007	Pearson Correlation	-.006	-.006	-.015	-.006	-.018	-.006	1	-.009	-.008	-.009	.786	.705	.838	.054	.635	.815	-.021	-.006	.609	-.006	-.008	.724	.537	-.009	-.011	-.006	-.006	.735	-.006	-.006
	Sig. (2-tailed)	.891	.891	.740	.891	.695	.891		.846	.853	.846	.000	.000	.000	.235	.000	.000	.652	.891	.000	.891	.853	.000	.000	.846	.811	.891	.891	.000	.891	.891
E008	Pearson Correlation	-.003	-.003	-.007	.703	-.008	.710	-.009	1	.321	-.004	-.011	-.009	-.009	-.007	-.009	-.008	-.010	.710	-.012	.710	-.004	-.009	-.012	-.004	-.005	-.003	.710	-.011	.710	-.003
	Sig. (2-tailed)	.949	.949	.877	.000	.855	.000	.846		.000	.927	.805	.840	.837	.873	.843	.855	.833	.000	.795	.000	.931	.837	.788	.927	.911	.949	.000	.813	.000	.949
E009	Pearson Correlation	-.003	-.003	.191	-.003	-.008	.453	-.008	.321	1	-.004	.074	-.009	-.009	.188	-.009	-.008	.592	.453	.151	.453	.791	-.009	-.012	-.004	.490	-.003	.453	.066	.453	.890
	Sig. (2-tailed)	.951	.951	.000	.951	.862	.000	.853	.000		.931	.105	.848	.845	.000	.851	.862	.000	.000	.001	.000	.000	.845	.799	.931	.000	.951	.000	.148	.000	.000
E010	Pearson Correlation	.726	.726	-.007	-.003	-.008	-.003	-.009	-.004	-.004	1	-.011	-.009	-.009	.436	.402	-.008	.141	-.003	.315	-.003	.330	-.009	-.012	-.004	.424	-.003	-.003	.113	-.003	-.003
	Sig. (2-tailed)	.000	.000	.877	.949	.855	.949	.846	.927	.931		.805	.840	.837	.000	.000	.855	.002	.949	.000	.949	.000	.837	.789	.927	.000	.949	.949	.013	.949	.949
E011	Pearson Correlation	-.008	-.008	.105	-.008	-.023	-.008	.786	-.011	.074	-.011	1	.867	.850	.266	.750	.848	.215	-.008	.868	-.008	.074	.880	.558	.271	.231	.317	-.008	.892	-.008	.087
	Sig. (2-tailed)	.862	.862	.021	.862	.620	.862	.000	.805	.105	.805		.000	.000	.000	.000	.000	.000	.862	.000	.862	.106	.000	.000	.000	.000	.000	.862	.000	.862	.056
E012	Pearson Correlation	-.007	-.007	-.016	-.007	-.019	-.007	.705	-.009	-.009	-.009	.867	1	.666	.279	.652	.843	.134	-.007	.772	-.007	-.009	.983	.505	.507	-.011	-.007	-.007	.896	-.007	-.007
	Sig. (2-tailed)	.887	.887	.731	.887	.684	.887	.000	.840	.848	.840	.000		.000	.000	.000	.000	.003	.887	.000	.887	.847	0.000	.000	.000	.804	.887	.887	.000	.887	.887
E013	Pearson Correlation	-.007	-.007	-.016	-.007	-.019	-.007	.838	-.009	-.009	-.009	.850	.666	1	.050	.694	.770	-.022	-.007	.671	-.007	-.009	.683	.557	-.009	.186	.327	-.007	.833	-.007	-.007
	Sig. (2-tailed)	.885	.885	.726	.885	.679	.885	.000	.837	.845	.837	.000	.000		.277	.000	.000	.634	.885	.000	.885	.845	.000	.000	.837	.000	.000	.885	.000	.885	.885
E014	Pearson Correlation	.605	.605	.036	-.005	-.015	-.005	.054	-.007	.188	.436	.266	.279	.050	1	.387	.064	.385	-.005	.570	-.005	.468	.238	.029	.411	.471	-.005	-.005	.324	-.005	.214
	Sig. (2-tailed)	.000	.000	.429	.910	.748	.910	.235	.873	.000	.000	.000	.000	.277		.000	.159	.000	.910	.000	.910	.000	.000	.524	.000	.000	.910	.910	.000	.910	.000
E015	Pearson Correlation	.558	.558	-.015	-.006	-.018	-.006	.635	-.009	-.009	.402	.750	.652	.694	.387	1	.775	.096	-.006	.883	-.006	.250	.690	.485	-.009	.491	.280	-.006	.751	-.006	-.006
	Sig. (2-tailed)	.000	.000	.736	.889	.691	.889	.000	.843	.851	.000	.000	.000	.000	.000		.000	.035	.889	.000	.889	.000	.000	.000	.843	.000	.000	.889	.000	.889	.889
E016	Pearson Correlation	-.006	-.006	-.014	-.006	-.017	-.006	.815	-.008	-.008	-.008	.848	.843	.770	.064	.775	1	-.019	-.006	.711	-.006	-.008	.890	.600	-.008	-.010	-.006	-.006	.751	-.006	-.006
	Sig. (2-tailed)	.897	.897	.755	.897	.712	.897	.000	.855	.862	.855	.000	.000	.000	.159	.000		.671	.897	.000	.897	.861	.000	.000	.855	.822	.897	.897	.000	.897	.897
E017	Pearson Correlation	.201	.201	.623	-.007	-.019	-.007	-.021	-.010	.592	.141	.215	.134	-.022	.385	.096	-.019	1	-.007	.403	-.007	.686	.115	.006	.291	.484	-.007	-.007	.208	-.007	.668
E018	Pearson Correlation	-.002	-.002	-.005	-.002	-.006	1.000	-.006	.710	.453	-.003	-.008	-.007	-.007	-.005	-.006	-.006	-.007	1	-.008	1.000	-.003	-.007	-.009	-.003	-.004	-.002	1.000	-.008	1.000	-.002
	Sig. (2-tailed)	.964	.964	.913	.964	.897	0.000	.891	.000	.000	.949	.862	.887	.885	.910	.889	.897	.881		.854	0.000	.951	.885	.850	.949	.937	.964	0.000	.867	0.000	.964
E019	Pearson Correlation	.441	.441	.152	-.008	-.024	-.008	.609	-.012	.151	.315	.868	.772	.671	.570	.883	.711	.403	-.008	1	-.008	.356	.776	.457	.309	.523	.283	-.008	.850	-.008	.173
	Sig. (2-tailed)	.000	.000	.001	.854	.601	.854	.000	.795	.001	.000	.000	.000	.000	.000	.000	.000	.000	.854		.854	.000	.000	.000	.000	.000	.000	.854	.000	.854	.000

Figure 6.5: A snapshot of a correlation matrix (generated by SPSS) of interpolated occurrence strength of event types.

6.3 Metric Selection

As discussed in Chapter 3, and similar to the rolling upgrade case study, in this section, we aim to find which of the metrics have the highest sensitivity to the log activities from the operation.

For this case study, we received middleware monitoring data collected from system probes by the loadable Kernel module (explained in Section 6.1). The monitoring data includes several metrics with timestamps of approximately 100-millisecond frequency. Table 6.1 shows the list of metrics of monitoring data that were available with timestamps.

It is worth noting that there were some other monitoring metrics included in the data of the case study such as Load Average and Network traffic-related metrics show sensitivity to the activities of operation. However, due to the lack of timestamps for monitoring records, we had to disregard them for the analysis. This is because timestamp is a pre-requisite for our analysis in order to map the log activities to the monitoring metric data.

As can be seen in Table 6.1, among the metrics that were available with timestamps, some of these could not be used for statistical analysis as they did not come with complete data: their values were filled with zero or just one constant value was recorded for all the metrics records. Therefore, we dropped these non-suitable metrics for our analysis, which are listed as *Not Valid* metrics in Table 6.1. As a result of this, we ended up having 11 metrics that were suitable for the statistical analysis.

Based on the above outcome, we used the monitoring metrics, along with the metrics derived from the log activities, and performed regression analysis to assess how well each target monitoring metric shows sensitivity to the activities reported in the logs. Similar to the rolling upgrade case study, the metrics from the log activities are taken as the predictor variables in our regression analysis, and each resource metric as a target metric. The result of applying regression is shown in Table 6.2.

Similar to what we had for the rolling upgrade case study, in the table, R denotes the correlation between a given monitoring metric and the occurrences of activities from the event logs, and R^2 indicates how well the model predicts

Table 6.1: List of available metrics

Metric	Validity Status
CPU usage - (Average of: CPU usage - core 0, CPU usage - core 1, CPU Usage - core 2, CPU usage - core 3)	Valid
number of voluntary context switches	Not Valid
number of involuntary context switches	Not Valid
RAM usage	Valid
VM current size	Valid
VM peak size	Valid
VM currently resident in RAM	Valid
peak of VM resident in RAM	Valid
VM size for data	Valid
VM size for stack	Not Valid
VM size for code	Not Valid
number of page faults	Valid
disk read	Not Valid
disk write	Not Valid
number of opened files	Valid
number of sockets	Valid
heap size	Valid

Table 6.2: Coefficient correlation and coefficient determination results for each metric

Metric	R	R^2	$Adj.R^2$	*p-value*
CPU Usage	0.877	0.769	0.761	0.000
RAM usage	0.198	0.039	0.008	0.218
VM current size	0.716	0.513	0.498	0.000
VM peak size	0.542	0.294	0.271	0.000
VM currently resident in RAM	0.186	0.034	0.003	0.0
Peak of VM resident in RAM	0.208	0.043	0.012	0.141
VM size for data	0.308	0.095	0.064	0.0
Number of minor page faults	0.273	0.074	0.045	0.000
Number of opened files	0.380	0.145	0.115	0.0
Number of sockets	0.381	0.145	0.116	0.000
Heap size	0.246	0.060	0.028	0.021

new observations. R^2 is used to assess the predictive power of the regression model for the given predictors and target variables [99]. $Adj.R^2$ is a modification version of R^2 that adjusts for the number of predictors in a model [99]. Prediction abilities of the metrics based on the value of $Adj.R^2$ are shown in Fig. 6.6.

By looking at Table 6.2 and Fig. 6.6, we observe the only metric that shows strong values of R^2 and $Adj.R^2$ is the *CPU usage* with values of 0.769 and 0.761 for R^2 and $Adj.R^2$, respectively. Second to CPU usage is *VM current size*, with a correlation coefficient of $R^2 = 0.513$ and $Adj.R^2 = 0.498$, which indicates not as strong correlation as for CPU usage. The rest of the metrics show almost none to fairly low correlation to the log activities.

Given the above outcome, CPU usage is the best metric candidate for using the log as the context, along with the metrics, for anomaly detection. The strong value for the CPU metric suggests that the variation of the CPU usage metric should be explained by our regression model and therefore, changes in values in the CPU that may not be predicted from the log activities may be the indicator of anomalies in a system. We will investigate this hypothesis in the next section.

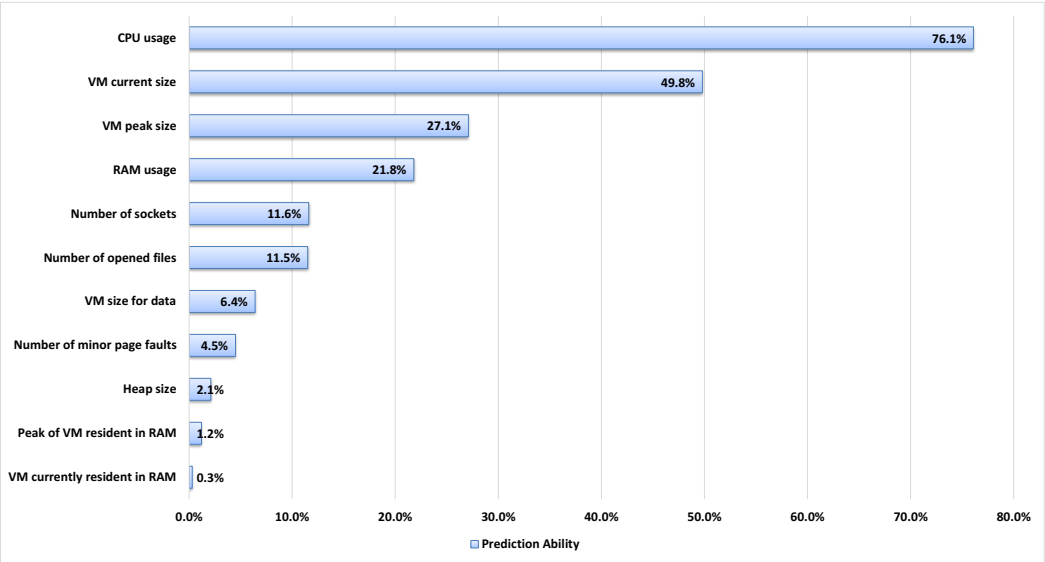


Figure 6.6: Prediction ability for each monitoring metric, based on $Adj.R^2$

6.4 Assertion Derivation

As mentioned before, one of the objectives of performing multiple regression analysis is to find an explanatory relationship between the independent variables (activities extracted from logs) and the dependent variables (monitoring metrics). In the previous section, we identified CPU usage as a good candidate metric, as it had high correlation with the log activities. In this section, we aim to find out which of the log activities are affecting the target metric (CPU usage), in order to derive assertion specifications for our anomaly detection.

In this direction and based on one of the research questions (RQ3) of this thesis, we would like to investigate the generalisability of our approach on identifying the influential log activities of this case study. Based on our findings with the first case study, we expect by employing the exploratory power of regression analysis we would be able to derive assertion specifications where the value of the metric at each time window could be predicted from log activities based on the derived assertion equation.

In order to perform such analyses, we take the regression coefficient for each predictive metric of our multiple regression models generated by the regression analysis, based on the steps discussed in Section 3.4 and similar to the one applied in the rolling upgrade case study. In this process, log activities are taken as input predictor variables and CPU usage as the target variable, and then the regression results are obtained[3]. The coefficient results for all predictors are shown in Table 6.3.

The key indicator for identifying predictors that do not have significance on a regression model is by looking at the *p-value*. Therefore, by checking the *p-value*s in Table 6.3, we observe that the coefficients of the activities A01, A02, A03, A04, and A09 are statistically insignificant ($p > .05$) [4]. These observations allowed us to narrow the set of contributing activities down to the 11 activities that are shown in Table 6.4.

Rerunning multiple regression with activities that have statistical significance ($p < .05$) resulted in the outcomes shown in Table 6.4. By looking at the

[3]We performed all our statistical analysis, including generating regression coefficient results, using IBM SPSS (https://www.ibm.com/analytics/au/en/technology/spss/).

[4]The p-value of 0.05 is commonly chosen as an acceptable level of significance [41, 98].

Table 6.3: Coefficients for identified influential factors

Predictors	β	Std. Error	B	*p-value*
Intercept (Constant)	1.480	0.104	—	0.000
A01	0.186	0.770	0.008	0.810
A02	-0.363	0.818	0.010	0.658
A03	-0.1.257	1.075	0.057	0.243
A04	-0.135	1.180	0.006	0.909
A05	4.975	2.231	0.085	0.260
A06	3.981	1.863	0.066	0.033
A07	19.799	1.091	0.405	0.000
A08	3.766	0.208	0.403	0.000
A09	-3.108	2.791	0.042	0.266
A10	-38.030	6.065	0.352	0.000
A11	4.368	1.970	0.086	0.027
A12	0.719	0.406	0.044	0.047
A13	11.157	2.348	0.106	0.012
A14	-14.192	5.904	0.145	0.001
A15	7.723	2.356	0.073	0.000
A16	48.403	5.112	0.000	0.000
A17	3.901	0.515	0.816	0.000

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

standardized coefficient values in Table 6.4, we can understand the predictive power of each log activity for CPU usage. Also, Fig. 6.7 shows the importance or relative contribution of each predictor for CPU usage.

In addition, we use the unstandardised coefficient (β) values from Table 6.4 to derive the assertion equation that can be used for the prediction of CPU usage at each second. The derived assertion equation is presented in Equation 6.1. The actual value is expected to be predicted based on this equation with the Standard Error (refer to page 55) of estimate of 6.059 (absolute value).

$$PredictedCPUUsage_i = 0.1.462 + 14.606 * A06_i + ... + 3.557 * A17_i \quad (6.1)$$

Table 6.4: Coefficient for identified influential factors

Predictors	β	Std. Error	B	*p-value*
Intercept (Constant)	1.462	0.101	—	0.000
A06	4.606	1.819	0.077	0.012
A07	19.808	1.819	0.405	0.000
A08	3.766	0.208	0.403	0.000
A10	-30.411	3.955	0.281	0.000
A11	2.539	1.128	0.050	0.025
A12	0.797	0.394	0.049	0.025
A13	11.176	2.344	0.106	0.044
A14	-9.358	3.699	0.091	0.000
A15	7.741	2.351	0.073	0.012
A16	41.415	2.890	0.698	0.000
A17	3.557	0.463	0.216	0.000

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

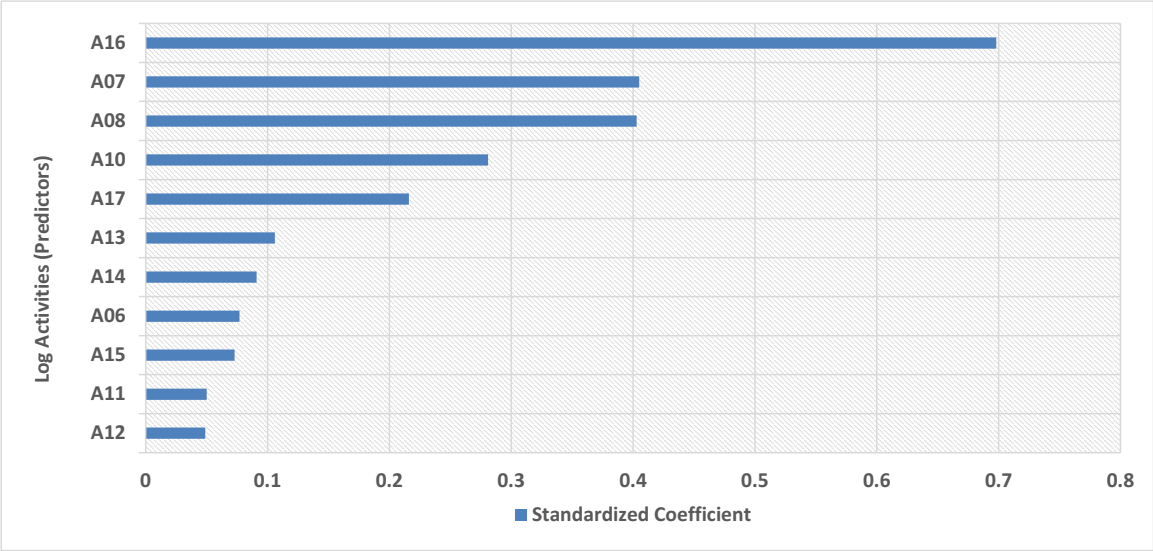


Figure 6.7: Prediction influence of each log activity on CPU usage, based on standardized regression coefficient(B) extracted from regression analysis

6.5 Anomaly Detection

Based on the analysis of the training dataset in the previous section, we managed to identify the most sensitive metric and derive the assertion equation. Also, we identified which activities in the log have the highest contribution to

the changes in CPU usage. In this section, we aim to use these findings and evaluate how these outcomes can be utilised for predicting actual values and ultimately leverage that for contextual anomaly detection. In this direction, first, we evaluate how well the selected metric, along with the assertion formula from the learning model, can predict the actual metric values. Then, we evaluate if the proposed approach is effective for detecting three types of emulated errors in the system.

It is worth restating that the author of this thesis had no influence on the experimental campaign of the testing data and the emulation of the errors that are presented in this section. Therefore, it was valuable for us to evaluate the generalisability of our approach on a case study obtained from an external source.

We obtained the raw data from the Dipartimento di Informatica e Sistemistica - Università degli Studi di Napoli Federico II. Accessing this data gave us the chance to evaluate the applicability and generalisability of the proposed approach of this thesis for a separate industry-grade case study which has a different environment and scale from the case study of the rolling upgrade that we presented in the previous chapter.

6.5.1 Predictability Power Evaluation

In this section, we aim to address the following question: having the activities reported in the logs, can we predict the values of the selected target metric using the derived assertion equation from the regression model?

In order to answer this question, we leverage the testing data sets from four runs of the FDP middleware: one with the training data itself labelled as Normal Run, and three others with the data that comes with the error emulated records.

To enable the collection of data under anomalous conditions, three types of errors were emulated into code pieces of FDP application; these three types of errors are as follows:

- Active Hang: The system appears to be running, but its services may

be perceived as unresponsive; in such a hang, CPU cycles are typically consumed uselessly.

- Passive Hang: The system appears to be running, but its services may be perceived as unresponsive, typically because of an indefinite wait for resources that will never be released within an expected time-out.

- Crash: The system terminates unexpectedly and is not able to execute subsequent method invocations.

The fault injection process in three runs that contain error emulated records has been instrumented in a way to activate and deactivate the error emulation during the execution of the experimental campaign. Therefore, we can track which data records are collected under the condition of fault injection and which ones under normal conditions.

Given the above explanation, in this subsection, we focus on understanding the predictability power of our approach for non-faulty records. In other words, for three data sets that contain errors, only the non-faulty records will be employed for the evaluation.

As with the normal run, we used the derived assertion equation formula from Equation 6.1, taking the occurrence of log activities as input and estimating the CPU usage at each time window. First, we applied the above equation and calculated the outcome for each record for the Normal Run. The results show that out of 483 records of data, 480 records have the predictions within the standard error of the estimate. Fig. 6.8.a depicts a line chart of the actual values versus predicted values, which the records on the horizontal axis indicate individual seconds.

The results obtained show that the model demonstrates an accuracy (i.e. trueness) of 99.4% on predicting CPU usage within the accepted threshold (error estimate of 6.059 (absolute)). The results are shown in Fig. 6.9.

So far the analysis shows that the prediction has a very high accuracy, but this assessment has been done on the data set that the training has been based on, and therefore there might be a chance of overly influencing the result obtained. In order to make sure that the model has effective predictability

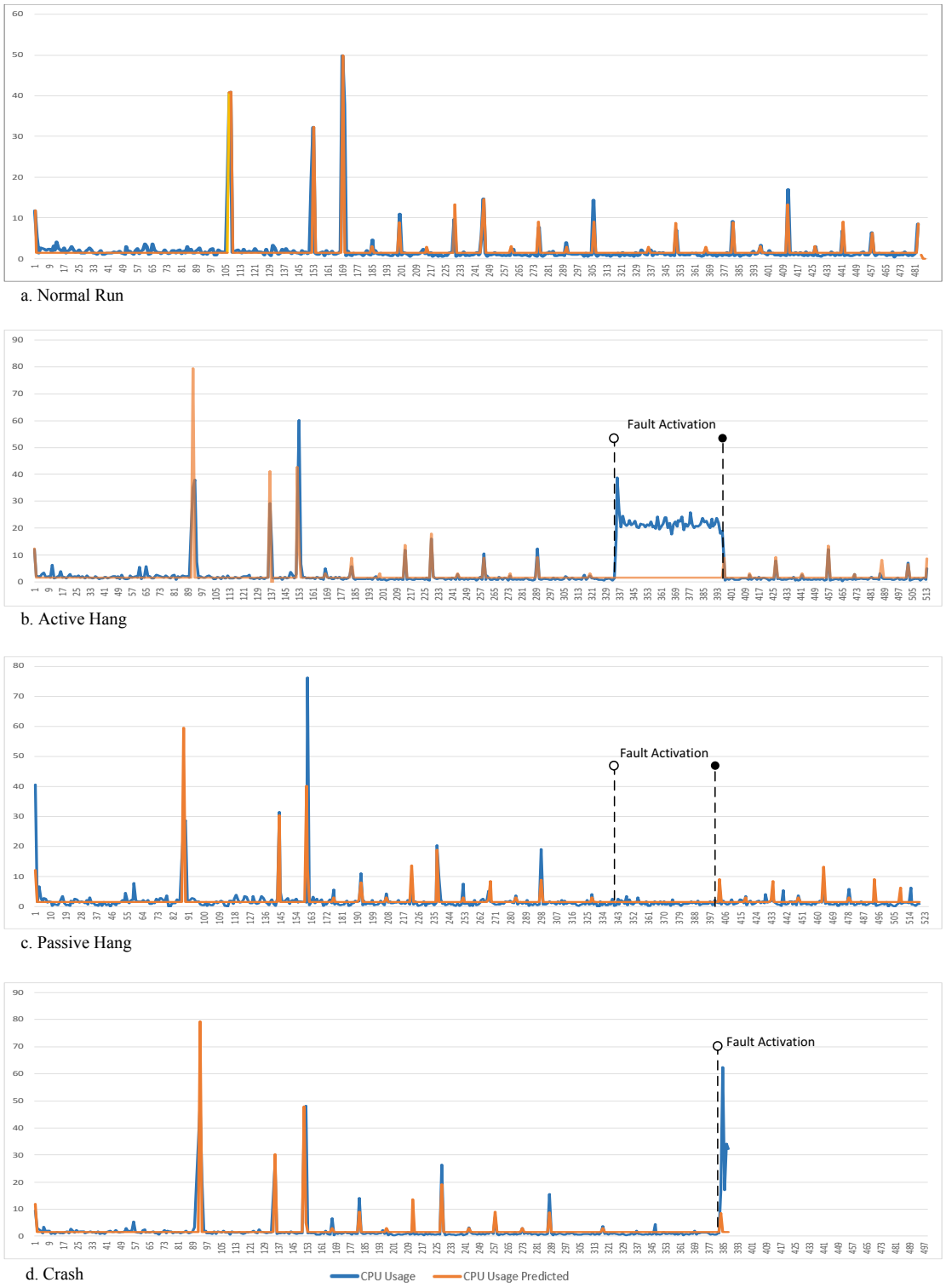


Figure 6.8: Actual CPU usage versus predicted CPU usage for four separate runs: Normal, Active Hang, Passive Hang, and Crash with highlighted fault activation periods where present.

for an out-of-sample data set, we evaluated the approach for three separate experiments, named Active Hang, Passive Hang and Crash. (Fault-injection details for each of these three experiments are described in the next section.)

Table 6.5: Accuracy of prediction of CPU usage in four different experiments

	Normal Run	Active Hang	Passive Hang	Crash
Total Records	483	513	519	387
Non-Faulty Records	483	451	459	383
True Predictions	480	443	448	372
False Predictions	3	8	11	11
Accuracy	0.994	0.982	0.976	0.971

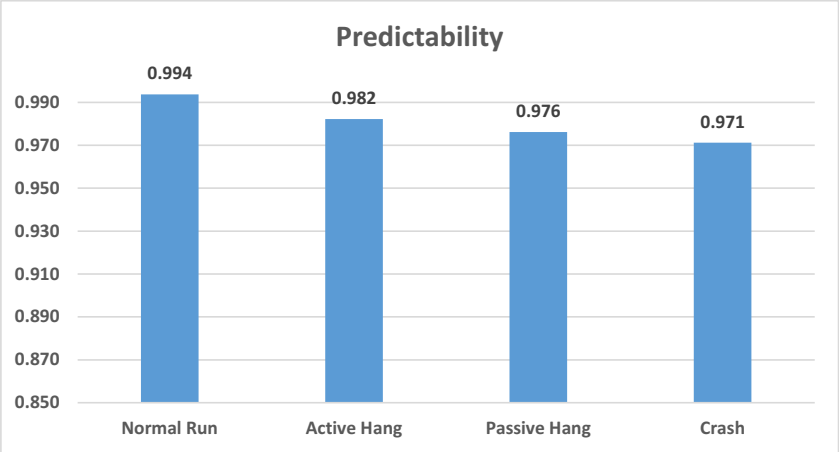


Figure 6.9: Actual CPU usage versus predicted CPU usage from assertion equation

For all these experiments, similar to the above, we obtained the counting of log activities for each experiment and simply employed the equation from Equation 6.1 to predict the CPU usage. In these experiments, the accuracy of prediction is solely calculated based on non-fault-injection records, which means the records during the fault-injection period were excluded from the analysis. Table 6.5 and line charts in Figs. 6.8 show the results of actual CPU usage in comparison with the predicted CPU usage.

The graphs in these figures demonstrate that the estimated CPU usage almost mimics the pattern of actual CPU usage, with a few cases of the wrong prediction for each experiment, which was reported in Table 6.5. Please note the gaps between actual values and predicted values that can be observed in records 334 to 337 of Active Hang in Fig. 6.8, and that the last records of Crash were part of the records with fault-injection, and were excluded, as in this section our sole focus was on gauging the predictability power of our

model. We will investigate the effectiveness of our approach to detect these anomalous cases in the next section.

Based on the results presented in Table 6.5, the proposed approach is shown to be effective in having high accuracy in correctly predicting CPU usage from the activities reported from the logs, and thus providing a positive answer to the research question we highlighted at the beginning of this section.

6.5.2 Anomaly Detection Evaluation

One of the main research questions of this thesis is whether the correlation learned from the activity of logs and resource metrics can be used for detecting system health and performance anomalies. In the previous section, we showed that the assertion equation formula derived from regression analysis was effective in predicting CPU usage in the non-faulty parts of four separate experimental runs. In this section, we aim to leverage this predictability power and evaluate the applicability of the proposed approach for detecting anomalies.

As we stated in the previous section, to enable the collection of data under anomalous conditions, three types of errors were emulated into code pieces of FDP application with Active Hang, Passive Hang, and Crash.

For each experiment, after starting the middleware, first the middleware is exposed to a nominal workload, then the error emulation is performed a few minutes after the beginning of the operation. The triggering of hangs for both Passive Hang and Active Hang were aborted automatically around one minute after the start of the error emulation, followed by a normal workload. However, for the experiment with the Crash error, the system went out of service after few seconds of error emulation.

In order to measure the precision and recall of the prediction, similar to the rolling upgrade case study, we classified the result of the prediction into four categories: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). Table 5.5 explains these four categories in terms of an alarm being raised (or not), and a fault is injected (or not). For any of the data records, we aim to raise an alarm when a fault is injected (TP) or raise

no alarm when no fault is injected (TN). FP and FN thus mark cases where the prediction did not work perfectly. These four categories are the basis for calculating precision, recall, and F-measure [83].

Anomaly Detection Result

Given the above description, and using the assertion equation, the anomaly detection is run and results are obtained. The difference between actual metric value and predicted values from regression output are shown in Figs. 6.8 for each error emulation run, respectively. Also, the precision, recall, and F-Score for each of the three experiments are presented in Table 6.6.

With Active Hang, we can observe in Fig. 6.8.b that the predicted values closely mimic the actual values. In a few points, such as in records 90-91 (horizontal axis in Fig. 6.8.b) and 153, there are considerable differences between actual values and predicted values. The predicted values follow a similar pattern (sharp increase on CPU usage) to the actual values.

At about record 334, a sudden significant gap between the actual value and predicted value appears and continues for a period of around 60 seconds. This time period is exactly the time during which the fault injections were activated. From this observation, we hypothesise anomalies as the result of differences of actual and predicted values to be detected with fairly good accuracy.

Looking at the results of the anomaly detection of Active Hang in Table 6.6 (for zero second time window - 0sTW) confirms our visual observation. In this process, out of 513 records of data, we had 443 cases of TN, 0 cases of FN, 62 cases of TP, and eight cases of FP. As a result, we have a precision of 0.886, recall of 1.000, and F-Score of 0.939. One the one hand, the recall value of 1.000 shows the proposed approach managed to successfully detect all of the Active Hang anomalies. On the other hand, there were few cases (92-94, 137-138, 153-154, 488) of false reported anomalies that affected the precision value.

As with the Passive Hang, similar patterns can be observed in Fig. 6.8.c, with the exception that there is no gap between prediction and actual value when the Passive Hang fault injection is activated from record 342. In this

Table 6.6: Anomaly detection results.

	Active Hang		Passive Hang		Crash	
Metric	0sTW	1sTW	0sTW	1sTW	0sTW	1sTW
Precision	0.886	0.925	0.000	0.000	0.267	0.333
Recall	1.000	1.000	0.000	0.000	1.000	1.000
F-Score	0.939	0.961	NA	NA	0.421	0.500

*Notes: 0sTW indicates Zero second time window and 1sTW indicate One second time window.

experiment, we obtained 519 records of data, where we had 450 cases of TN, 58 cases of FN, 0 cases of TP, and 11 cases of FP.

Having 0 cases of TP (zero detection of anomalies) results in precision and recall of 0, clearly indicating that the approach is ineffective for detecting Passive Hang errors. This is an important observation. During a passive hang, the system goes to an indefinite waiting state for resources, causing a pause on operation activities. As such, no logs are emitted and no changes can be observed in resource metrics (with a mean close to inactivity) during this time. In other words, this type of error is asymptomatic in both log activities and CPU usage, meaning that the fault does not affect CPU usage and does not leave log traces in the log file.

The experiment run with the Passive Hang revealed an important limitation in our approach. We employ both logs and metrics, and having anomalous symptoms on each of these sources of monitoring data can lead to the detection of anomalies. Nevertheless, we cannot find errors that impact both simultaneously. It is worth noting that, POD-Detection that focuses on error detection among others by process conformance checking [131, 140], has the ability to detect timing anomalies in the log behaviours. With POD-Discovery, in cases that logs are not reported within the range of expected frequency, timing anomalies will be reported. Therefore, the limitation of our approach can be covered by employing POD-Discovery along our anomaly Checker.

The last experiment contains the error emulation that caused the system to crash. In this experiment, the prediction fairly resembles the actual values where there is normal run, however, after the activation of error, the operation execution is aborted in few seconds - see Fig. 6.8.d. In this process, out of

387 records of data we had 372 cases of TN, 0 cases of FN, four cases of TP, and 11 cases of FP. As a result, we have a precision of 0.267, recall of 1.000, and an F-Score of 0.421. The recall value shows that the approach successfully managed to detect the anomalies that occurred, however, the low precision value is an indicator of the existence of too many false alarms in comparison to true positive alarms.

So far the results presented were based on a default time window of zero second. In addition to the above, and similar to the rolling upgrade case study, we expanded our time window of analysis to ±1 Second. As a result, the number of false positives slightly reduced and the precision improved. Additional expansion of time window to ±2 seconds did not lead to improvement of the result. Table 6.6 shows the results of a zero-second and one-second time window.

Anomaly Detection with Change Detection

We learned from the analysis in the previous section, that the predicted values mimic the same pattern of actual values, however, in some cases, it can not correctly predict the hight of the spikes. For example, in Fig, 6.8.c record 160 shows a peak in CPU usage both in predicted value and actual value. However, the difference between these two values is fairly large, as the actual value indicates 76.16% CPU usage while the predicted value indicates 40.14% CPU usage. Both of these values are far bigger than the mean value (2.05) of CPU usage. This implies that both of these high values for CPU usage are very likely to be caused by the activities reported in the logs, but still our approach detected them as anomalies. This is because the criterion for detecting anomalous instances we had used so far was based on the gap between the predicted value and the actual value.

The problem of modelling workload bursts or spikes in system monitoring, resource allocation and anomaly detection have been a topic of interest in recent studies [16, 87, 111]. This inspired us to investigate further whether we can improve the accuracy of our prediction from the perspective of change detection along with the prediction obtained from our assertion equation. To

Algorithm 2 - Change Detection

- Anomaly Detection:
1: **if** $(|ac - pr| < \epsilon)$ **then**
2: $anomaly \leftarrow false$
3: **else if** $(|ac - m| < \sigma$ AND $|pr - m| < \sigma)$ OR $(ac > m + \sigma$ AND $pr > m + \sigma)$ OR $(ac < m - \sigma$ AND $pr < m - \sigma)$ **then**
4: $anomaly \leftarrow false$
5: **else**
6: $anomaly \leftarrow true$
7: **end if**

achieve this, we propose a new threshold policy for detecting anomalies. In this new policy, we check whether both predicted and actual values indicate a significant change from the mean. The algorithm underlying the proposed policy is described in Algorithm 2, here referred as the *Change Detection* algorithm.

Let us denote the actual value as ac, the predicted value as pr, standard deviation as σ, and standard error of estimate as ϵ. The Change Detection algorithm first checks whether the difference between ac and pr is within the accepted error of estimate, i.e., ϵ. If the condition is not satisfied then instead of reporting the record as an anomalous instance, it checks if both ac and pr have similar changes with respect to the standard deviation σ from the mean m. Otherwise, it reports the record as an anomaly. It worth to note that in most statistical based anomaly detection techniques, standard deviations (σ) from the mean are used as a threshold to detect significant deviation from normal behavior; often the values larger than $\pm 3.0\sigma$ are considered outliers [102]. The use of tighter thresholds than in the literature becomes possible in our setting, since we gain additional precision from analyzing the log context.

By employing the change detection algorithm, we managed to reduce the number of false positives for all three experiments. The number of false positives reduced from eight to four, 11 to eight, and 11 to six cases for Active Hang, Passive Hang, and Crash, respectively. The precision, recall, and F-Score with the change detection algorithm with zero-second time window and one-second time window along with previous results are shown in Table 6.7.

Recalling the research question highlighted at the beginning of this section, the above results demonstrate that the proposed approach of this thesis was effective in modelling the relationship between log activities and resource metrics. A derived assertion equation from a regression model was shown, which

can be employed to predict the influence of the FDP application operation reported in logs on the CPU usage metric with high predictability power. In addition, the results show that the proposed approach is highly effective for detecting two of three types of anomalies emulated in the system. Further, we observed a significant improvement in precision by employing the proposed change detection technique and slightly expanding the time window of the observation. Figs. 6.10, 6.11, and 6.12 show the comparative view of precision, recall and F-score for all the conditions.

Table 6.7: Anomaly detection results with Change Detection.

	Active Hang				Passive Hang				Crash			
Metric	0sTW	1sTW	0sTW-CD	1sTW-CD	0sTW	1sTW	0sTW-CD	1sTW-CD	0sTW	1sTW	0sTW-CD	1sTW-CD
Precision	0.886	0.925	0.939	1.000	0.000	0.000	0.000	0.000	0.267	0.333	0.400	0.800
Recall	1.000	1.000	1.000	1.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000	1.000
F-Score	0.939	0.961	0.969	1.000	NA	NA	NA	NA	0.421	0.500	0.571	0.889

*Notes:
0sTW = Zero second time window.
1sTW = One second time window.
0sTW-CD = Zero second time window with Change Detection.
1sTW-CD = One second time window with Change Detection.

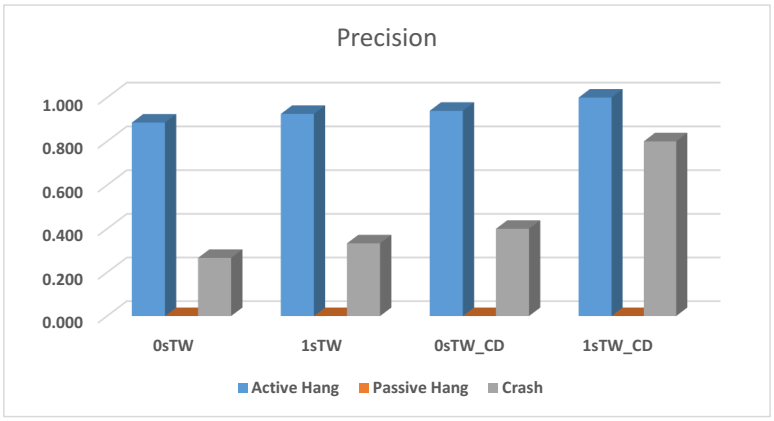


Figure 6.10: Precision of anomaly detection for three experiments with zero second time window, and with zero and one second time window with change detection

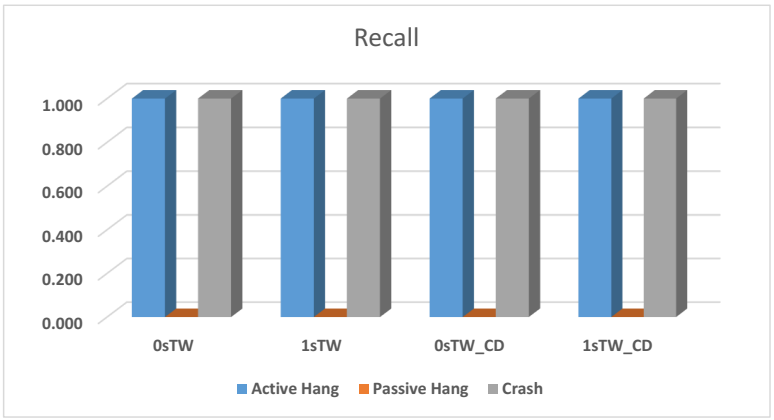


Figure 6.11: Recall of anomaly detection for three experiments with zero second time window, and with zero and one second time window with change detection

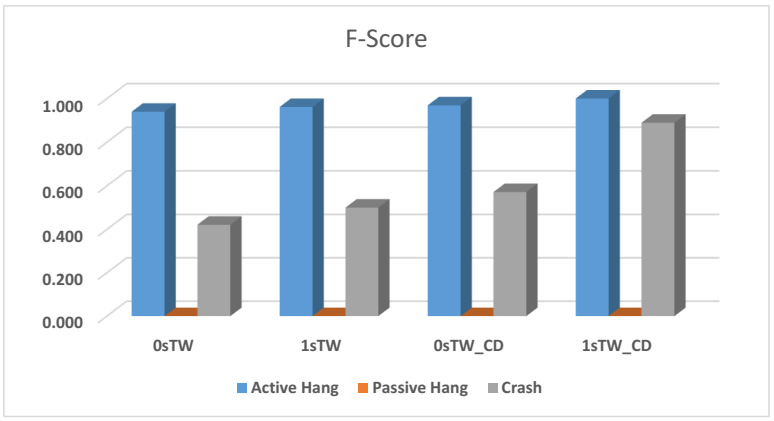


Figure 6.12: F-Score of anomaly detection for three experiments with zero second time window, and with zero and one second time window with change detection

6.6 Summary

In this chapter, we investigated the applicability and generalisability of the proposed approach of this thesis in an industrial flight data processor(FDP) case study. In addition, the case study discussed in the chapter has significant differences from the previous case study of the rolling upgrade from multiple aspects including the environment, the scale of logs and metrics, the forms of metrics available, and the type of faults injected.

Given FDP case study was from a different deployment environment, had a finer monitoring sampling rate, and a larger number of log event types, we did not observe that these key differences would affect the applicability or the accuracy of the predicted results. We learned that we can use the POD-Discovery tool to facilitate the log event type extraction process. Also, we found that employing the change detection technique can improve the anomaly detection rate of our approach. Moreover, we understood that our approach is mainly effective when the anomalies are not asymptomatic in logs and metrics. In other words, our approach is ineffective for those errors which simultaneously suppress the normative system activity and metric changes. This is a valuable observation that when the anomalous symptoms are absent at logs and metrics simultaneously, our approach has limitation to detect that.

Overall, we revisited the research questions of this thesis, and demonstrated that the proposed approach of this thesis is effective for mapping logs and metrics, identifying the most sensitive metrics for system monitoring, predicting the value of monitoring metrics from log activity with high accuracy, and finally, adopting the assertion specification for context-based anomaly detection of the system.

Chapter 7

Conclusions and Future Directions

This chapter provides the final conclusions for the work presented in this thesis. We summarise the main results and the contributions to the research questions of this thesis. Following that, we discuss potential improvements and extensions of the work described and outline future research that could be undertaken.

7.1 Summary

The major objective of this thesis was to investigate mechanisms to improve the dependability of system application operations using system monitoring and anomaly detection techniques. In particular, this thesis has focused on leveraging the dynamics of system application behaviour from logs, along with the state of resources from resource metrics, in order to perform metric selection and anomaly detection.

In this direction, we formulated a framework that includes the steps of parsing log messages and tracking the occurrence of log events, mapping logs to metrics, identifying the most relevant resource metrics and deriving assertion specifications for anomaly detection. In summary, our main contributions are in the following areas in this thesis.

As log events and resource metrics have different data formats [95] and they belong to two different sources of information, in our work we proposed an approach to map log events to resource metrics. For this objective, we

provided a solution using regular expressions to extract unique log event types, and counted the occurrence of log events at each time window, clustered low-granular logs to a set of log activities using weight timing and correlation analysis, and mapped log activities to resource metrics from the derived metrics from both logs and metrics.

Next, we proposed a systematic approach to identify the most sensitive resource monitoring metrics for anomaly detection. In particular, we showed that not all the metrics are affected by an application operation, and we also showed how a metric with higher sensitivity to operational behaviour can lead to a better anomaly detection rate. The issue we addressed here is particularly important, as dealing with too many monitoring metrics has been highlighted as one of the issues in system health and performance monitoring in several past studies [23, 60].

We proposed a method to derive assertion specification from the regression analysis. We showed how the result of a derived assertion equation can be employed to check whether the expected state of the system from observed activities in logs matches the actual resource metric data. The output of assertion derivation also helped us to understand what factors contribute the most to the changes in resources.

Further, a ripple effect detection mechanism is proposed in this thesis that has been shown to be helpful in distinguishing actual anomalies from the ripple effect of anomalies. The result of this effort contributes to the reduction of the excessive alarm rate that has been reported as one of the issues for today's system monitoring and anomaly detection approaches [94, 109, 138].

Finally, we explored and evaluated the proposed approach of this thesis with two different industry-grade case studies. We developed a prototype tool to conduct the experiments and we showed that the proposed approach is effective for detecting anomalies.

The main observations of investigating our approach to these two case studies are as follows:

- As the proposed approach is a non-intrusive approach, we showed that in both case studies, this approach could be implemented without the need

for instrumenting application code or the native environment to execute the operation and monitoring solutions.

- We showed that the proposed approach is applicable for different environmental settings. We demonstrated the applicability of the approach with a rolling upgrade case study running on Amazon Cloud Services with a well-established monitoring service, as well as with a flight data processor case study hosted on private data centres using native operating system monitoring solutions.

- One important difference between the two case studies is in the sampling rate monitoring of the metrics collection. We investigated a rolling upgrade operation based on one-minute monitoring intervals and an FDP case study based on a one-second time interval. Our approach was effective in both case studies for detecting the majority of anomalies.

- As our approach is designed to be unsupervised, we demonstrated with both case studies that the proposed approach was effective in developing a statistical model that can be used for anomaly detection for non-labelled data.

- From the analysis of case studies, we found that among the anomalies investigated, our approach is effective for detecting anomalies that show changes in behaviour of logs or values of resource metrics. If an anomaly remains asymptomatic (not affecting values of resource and log event patterns simultaneously), our approach will be ineffective. Also, our approach is not designed to address the anomalies solely affecting log events. However, POD-Detection [131, 140] has the ability to detect timing anomalies in the log behaviours and thus this limitation can be covered by employing POD-Discovery along side our Anomaly-Checker. Given this limitation, our approach managed to detect the type of anomalies that have impacts on metric resources, with high accuracy in both case studies.

Overall, the evaluation results of our experiments show that the proposed unsupervised context-based anomaly detection approach can be effectively

used to detect anomalies using analysis of logs and metrics in a non-intrusive way.

7.2 Answers to Research Questions

The research questions under consideration, as outlined in Chapter 1, can be answered based on the results of the work presented in this thesis. Below we provide concise responses to each of the research questions investigated:

Research Question 1:

As event logs and resource metrics are two separate sources of monitoring information with different structures, how can we combine the information from these two sources and derive a statistically meaningful relationship between them?

In order to address this research question, we proposed a set of steps to map the two heterogeneous data types of textual log messages and numerical metrics. From the log processing perspective, the first challenge we faced was related to parsing log messages and tracking the occurrence of log events and representing them as a quantitative metric. We showed in our experimental analysis that by employing regular expressions we can manage to parse log messages, identify unique log events, trace the recurrence of log events, and derive a metric that shows the interpolated occurrence strength of each event. Once we obtained the quantitative metrics from logs, we were able to map log activities to resource monitoring metrics based on their timestamps. This mapping helped us to perform statistical analysis on logs and metrics (as demonstrated in Chapters 5 and 6) and thus leverage the mapping as the basis of our work for addressing the next two main research questions.

Research Question 2:

What is an appropriate mechanism to distinguish insensitive monitoring metrics from the ones that are sensitive to the activities of system application

operations and how do we best leverage this to identify the most suitable monitoring metrics for anomaly detection?

To address this question, we proposed adopting a regression-based correlation analysis technique to identify a subset of metrics that have a statistically significant correlation with the activities of application operations. We explained this approach in Chapter 3 and investigated and evaluated it in two case studies in Chapters 5 and 6. The results of our analysis showed that the proposed metric selection approach helped to narrow down the metrics dimension to the most relevant metrics. Further, the selected metrics were shown to be effective to be used as the basis for system monitoring for anomaly detection, respectively.

Research Question 3:

How can we build a statistically supported model based on log and resource metrics data to derive assertions that enable high-accuracy, unsupervised, contextual anomaly detection?

In order to address this research question, we employed the solutions proposed for addressing RQ1 and RQ2 and having the above outcomes and using regression analysis, we showed that the relationship between the activities on logs and the changes in resources can be statically modelled. By taking advantage of both, the predictive and exploratory features of regression analysis, we found how well the statistical model can predict the outcome, as well as what factors (log activities), have the highest impact on changes in resource metrics. This helped us to derive assertion equations for anomaly detection. By adopting assertions at run-time, we demonstrated with two different case studies that the proposed approach was successful in detecting symptomatic anomalies with high accuracy. Our approach, however, has a limitation in finding the types of anomalies that are asymptomatic in both logs and metric simultaneously such as Passive Hang in FDP case study, which had no impact on resource usage and log behaviour.

7.3 Open Problems and Future Work

The work presented in this thesis can be extended and continued in few research directions. In particular, in the future, we would like to address some of the limitations we faced throughout the investigation with our work.

7.3.1 Optimal Time Window

An aspect that we wanted to investigate further is related to the monitoring sampling rate. What is the appropriate time window for detecting anomalies? Currently, we define our monitoring time window based on the availability of monitoring data and our intuition about the system environment to find an appropriate size for default time window for detecting anomalies.

Having a small time window may lead to issuing too many false alarms too soon as the effect of an operation activity or failure may not yet be fully reflected on resources. In contrast, expanding the time window causes a delay on detection of anomalies. This is a case-specific trade-off: how much more delay is acceptable for what kind of an increased accuracy of anomaly detection? Therefore, an interesting research question for future work is how to systematically find the suitable time window when the availability of monitoring data is not the limiting factor.

7.3.2 Automatic Error Diagnosis and Self-Healing

The focus of this thesis was on anomaly detection, and thus we did not explore error debugging and diagnosis when an anomaly is detected. We suggest that one of the interesting future research of our research is with regards to error debugging and diagnosis because with the help of the assertion specification derived in our analysis, we can extract which log activities and metrics were involved in the incidence of an anomaly. This knowledge of logs and metrics is likely to be helpful in debugging errors and providing suggestions as to what might be the potential causes. Especially, if error debugging will be assisted with the identification of the type of anomalies may assist this process. An extension of the above direction could be to design a self-adaptive operation.

Such an operation would be able to perform self-healing actions after an error occurs.

Another future area of research in extending the above is how we can distinguish the type of anomalies based on the symptoms and lasting impacts of the anomalies on the monitoring data. For instance, distinguishing ones that have short impact symptoms from the ones that have long impact symptoms, respectively. Also, can we use the knowledge derived from the detected type of anomalies and their impact on system resources to tag the detected anomalies with a severity level?

7.3.3 Best-Practices in Statistical-Based Anomaly Detection

Statistical techniques for data analysis is often considered to be one of the most robust techniques. Nevertheless, one of the limitations of our approach, similar to the most statistical based techniques, is that there is a degree of subjective analysis needed in the learning phase to derive a predictive model. One aspect in regards to this matter is related to the presence of anomalies in the training data. "Anomaly detection techniques typically assume that anomalies in data are rare when compared to normal instances, though this assumption is generally true, anomalies are not always rare" [23]. This applies to our approach as we mentioned throughout this thesis that in the learning process we assumed the occurrence of anomalous data instances are either absent or they occurrence is far less than the normal data instances. As a rule of thumb a presence of 5% outliers for anomalous instances do not have a significant effect on regression analysis, but what if the presence of anomalies to be more than this value? In case of presence of more percentage of anomalous instances, how that may affect the accuracy of the derived model? An investigation and guideline on this matter might be helpful.

7.3.4 Usage of Machine Learning Techniques

In this thesis, we employed statistical-based techniques. An alternative solution using machine learning technique for modelling the correlation between log activities and source metrics can be an interesting area for the future work. Machine learning and statistics have a shared ground yet a comparative study that reveals the benefits and drawbacks of each approach is worth investigating. In particular with the perspective of automating the steps of the learning phase.

7.3.5 Dynamic Reconfiguration Using Logs and Metrics Analysis

In addition, having a correlation model for logs and metrics can potentially be used in various research areas. An interesting research direction could be to adopt the correlation of log activities with resource metrics for the dynamic reconfiguration of systems. For example, knowing that certain operation activities put significant stress on CPU consumption, a predictive model can be learned and implemented for automatic scaling down and scaling up of the computing power of the system under load in the cloud computing environment.

Another aspect that the correlation between logs and metrics can be adopted for is dynamic monitoring solutions. We reported in our related work that system monitoring with a large number of resources is an intensive and costly process. A predictive statistical model has the potential to be used for dynamic switching from high-level monitoring to detailed monitoring and vice versa. A practical solution in this direction can bring value to the domain of system performance monitoring

7.4 Final Remarks

In this thesis, we addressed the problem of monitoring system application operations through log and metrics analysis. Our contributions include a novel

approach that assists in finding the subset with the most relevant monitoring metrics. It further includes employing those metrics for improving the dependability of the correct execution of system application 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 the event logs and resource metrics of operations. 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. We evaluated our approach with two separate industry-grade case studies. Our results demonstrate that our regression-based analysis technique was able to detect the emulated anomalies with high precision and recall, respectively.

Bibliography

[1] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[2] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, 2013.

[3] M. Agyemang, K. Barker, and R. Alhajj, "A comprehensive survey of numeric and symbolic outlier mining techniques," *Intelligent Data Analysis*, vol. 10, no. 6, pp. 521–538, 2006.

[4] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, May 2015.

[5] E. Aleskerov, B. Freisleben, and B. Rao, "Cardwatch: a neural network based database mining system for credit card fraud detection," in *Proceedings of the IEEE/IAFE Computational Intelligence for Financial Engineering (CIFEr)*, Mar 1997, pp. 220–226.

[6] Amazon Team, *Amazon Web Services(AWS) Documentation*, 2017. [Online]. Available: https://aws.amazon.com/documentation/

[7] Avaya, "Network downtime results in job, revenue loss," Mar. 2014. [Online]. Available: http://www.avaya.com/usa/about-avaya/newsroom/news-releases/2014/pr-140305/

[8] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts

and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[9] AWS User Guide, *Best Practices: Managing and Deploying Apps and Cookbooks*, 2013.

[10] T. Baier and J. Mendling, "Bridging abstraction layers in process mining by automated matching of events and activities," in *Proceedings of the 11th International Conference on Business Process Management (BPM)*, 2013, pp. 17–32.

[11] Z. A. Bakar, R. Mohemad, A. Ahmad, and M. M. Deris, "A comparative study for outlier detection techniques in data mining," in *Proceedings of the Cybernetics and Intelligent Systems*. IEEE, 2006, pp. 1–6.

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

[13] K. Beiske. (2017) Discovering the need for an indexing strategy in multi-tenant applications. [Online]. Available: https://www.elastic.co/blog/found-multi-tenancy

[14] K. Bhaduri, K. Das, and B. L. Matthews, "Detecting abnormal machine characteristics in cloud infrastructures," in *Proceedings of the11th IEEE International Conference on Data Mining*, 2011, pp. 137–144.

[15] R. Birke, I. Giurgiu, L. Y. Chen, D. Wiesmann, and T. Engbersen, "Failure analysis of virtual and physical machines: Patterns, causes and characteristics," in *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun 2014, pp. 1–12.

[16] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, 2010, pp. 241–252.

[17] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proceedings of the 5th European Conference on Computer Systems*. ACM, 2010, pp. 111–124.

[18] R. P. J. C. Bose and W. M. P. van der Aalst, "Discovering signature patterns from event logs," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, Apr 2013, pp. 111–118.

[19] W. Budgaga, "A framework for real-time, autonomous anomaly detection over voluminous time-series geospatial data streams," PhDThesis, Colorado State University, 2014.

[20] R. N. Calheiros, K. Ramamohanarao, R. Buyya, C. Leckie, and S. Versteeg, "On the effectiveness of isolation-based anomaly detection in cloud data centers," *Concurrency and Computation: Practice and Experience*, pp. 41–69, 2017.

[21] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya, "Optimal multiserver configuration for profit maximization in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1087–1096, Jun 2013.

[22] D. Cappuccio, "Ensure cost balances out with risk in high-availability data centers," Gartner, Tech. Rep., Jul. 2013. [Online]. Available: http://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime

[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 15:1–15:54, 2009.

[24] R. N. Charette, "Why software fails," *IEEE Spectrum*, vol. 42, no. 9, pp. 42–49, Sep 2005.

[25] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Anomaly? Application change? Or workload change? Towards automated detection

of application performance anomaly and change," in *Proceedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, Jun 2008, pp. 452–461.

[26] M. Cinque, R. D. Corte, and S. Russo, "Error monitoring for legacy mission-critical systems," in *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, Jun 2016, pp. 66–71.

[27] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, Jun 2013.

[28] M. Cinque, D. Cotroneo, R. D. Corte, and A. Pecchia, "What logs should you look at when an application fails? insights from an industrial case study," in *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, (DSN)*, Jul 2014, pp. 690–695.

[29] M. Cinque, D. Cotroneo, R. Natella, and 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)*, Jul 2010, pp. 457–466.

[30] R. J. Colville and G. Spafford, "Configuration management for virtual and cloud infrastructures," May 2013. [Online]. Available: http://goo.gl/P5edKP

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

[32] Data61, "Dependable cloud - make cloud use and operations more dependable," 2017. [Online]. Available: https://research.csiro.au/data61/wp-content/uploads/sites/85/2016/08/AAP-Overview.pdf

[33] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th International Conference on Autonomic Computing*. ACM, 2012, pp. 191–200.

[34] T. Dumitras and P. Narasimhan, "Why do upgrades fail and what can we do about it?" in *Proceedings of the 10th ACM/IFIP/USENIX International Middleware Conference*, Nov 2009, pp. 349–372.

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

[36] M. Farshchi, J.-G. Schneider, I. Weber, and 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)*, Nov. 2015, pp. 24–34.

[37] ——, "Metric selection and anomaly detection for cloud operations using log and metric correlation analysis," *Journal of Systems and Software*, pp. –, 2017.

[38] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives," *Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2918–2933, 2014.

[39] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at Facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, 2013.

[40] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transaction on Software Engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[41] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer, 2001.

[42] M. Fu, L. Zhu, I. Weber, L. Bass, A. Liu, and X. Xu, "Process-oriented non-intrusive recovery for sporadic operations on cloud," in *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun 2016, pp. 85–96.

[43] Q. Fu, J. G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proceedings of the Ninth IEEE International Conference on Data Mining*, Dec 2009, pp. 149–158.

[44] S. Fu, "Performance metric selection for autonomic anomaly detection on cloud computing systems," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, Dec 2011, pp. 1–5.

[45] J. Gaudreau, *VM Right-Sizing Best Practice Guide*, 2014.

[46] D. Gibson, "Six IT predictions for 2016," 2016. [Online]. Available: https://blog.varonis.com/tag/predictions/

[47] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *Journal of Artificial Intelligence Research*, vol. 46, no. 1, pp. 235–262, Jan. 2013.

[48] J. Gray, "Why do computers stop and what can be done about it?" in *Proceedings of the Symposium on reliability in distributed software and database systems*. Los Angeles, CA, USA, Jun. 1986, pp. 3–12.

[49] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems*, Sep 2013, pp. 205–214.

[50] Q. Guan, "Autonomic failure identification and diagnosis for building dependable cloud computing systems," PhDThesis, 2014.

[51] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and 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*, Nov 2014, pp. 7:1–7:14.

[52] N. J. Gunther, *Analyzing Computer System Performance with Perl::PDQ*. Springer Science & Business Media, 2011.

[53] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, "Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems," in *Proceedings of the 35th IEEE Symposium on Reliable Distributed Systems (SRDS)*, Sep 2016, pp. 51–60.

[54] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2016, pp. 654–661.

[55] F. Hermenier, "Virtualized hosting platforms," Project-Team OASIS, Tech. Rep., 2012. [Online]. Available: https://team.inria.fr/DAESD/files/2012/05/Fabien-BtrPlace.pdf

[56] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, Oct 2004.

[57] M. Hütterman, *DevOps for Developers*. Apress, 2012.

[58] L. Huang, X. Ke, K. Wong, and S. Mankovskii, "Symptom-based problem determination using log data abstraction," in *Proceedings of the Advanced Studies on Collaborative Research*, 2010, pp. 313–326.

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

[60] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–35, 2015.

[61] Infonetics, "The cost of server, application, and network downtime," Infonetics Research, Tech. Rep., Jan. 2015. [Online]. Available: https://goo.gl/Pa9Y0y

[62] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula, "STAR: self-tuning aggregation for scalable monitoring," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, Sep 2007, pp. 962–973.

[63] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *Proceedings of the 8th International Conference on Quality Software*, Aug 2008, pp. 181–186.

[64] K. R. Joshi, G. Bunker, F. Jahanian, A. van Moorsel, and J. Weinman, "Dependability in the cloud: Challenges and opportunities," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems Networks*, Jun 2009, pp. 103–104.

[65] H. Kang, X. Zhu, and J. L. Wong, "Dapa: Diagnosing application performance anomalies for virtualized infrastructures," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. USENIX Association, 2012, pp. 1–8.

[66] S. Kanj, F. Abdallah, T. Denœux, and K. Tout, "Editing training data for multi-label classification with the k-nearest neighbor rule," *Pattern Analysis and Applications*, vol. 19, no. 1, pp. 145–161, Feb 2016.

[67] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A self-adaptive hierarchical monitoring mechanism for clouds," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1029–1041, 2012.

[68] S. Kavulya, K. R. Joshi, F. D. Giandomenico, and P. Narasimhan, "Failure diagnosis of complex systems," in *Resilience Assessment and Evaluation of Computing Systems*. Springer, 2012, pp. 239–261.

[69] T. Kelly, "Transaction mix performance models: Methods and application to performance anomaly detection," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*. ACM, 2005, pp. 1–3.

[70] V. Kumar, "Parallel and distributed computing for cybersecurity," *IEEE Distributed Systems Online*, vol. 6, no. 10, 2005.

[71] M. H. Kutner, C. Nachtsheim, and J. Neter, *Applied linear regression models*. McGraw-Hill/Irwin, 2004.

[72] Laerd Statistics, "Pearson product-moment correlation," 2017. [Online]. Available: https://statistics.laerd.com/

[73] Z. Lan, Z. Zheng, and Y. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 21, no. 2, pp. 174–187, 2010.

[74] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: Supervised or unsupervised?" in *Proceedings of the 13th International Conference on Image Analysis and Processing (ICIAP)*, Sep 2005, pp. 50–57.

[75] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'01)*, 2001, pp. 130–143.

[76] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the 28th Australasian Conference on Computer Science*. Australian Computer Society, Inc., 2005, pp. 333–342.

[77] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.

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

[79] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of devops," in *Proceedings of the 16th International Conference on Agile Processes in Software Engineering and Extreme Programming: (XP).* Springer International Publishing, May 2015, pp. 212–217.

[80] J. P. Magalhaes and L. M. Silva, "Detection of performance anomalies in web-based applications," in *Proceedings of the 9th IEEE International Symposium on Network Computing and Applications*, Jul 2010, pp. 60–67.

[81] J. P. Magalhães and L. M. Silva, "Anomaly detection techniques for web-based applications: An experimental study," in *Proceedings of the 11th IEEE International Symposium on Network Computing and Applications*, Aug 2012, pp. 181–190.

[82] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2009, pp. 1255–1264.

[83] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel, "Performance measures for information extraction," in *Proceedings of DARPA broadcast news workshop*, 1999, pp. 249–252.

[84] L. Mariani and F. Pastore, "Automated identification of failure causes in system logs," in *Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2008, pp. 117–126.

[85] M. Markou and S. Singh, "Novelty detection: a review - part 1: statistical approaches," *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[86] ——, "Novelty detection: a review - part 2: neural network based approaches," *Signal Processing*, vol. 83, no. 12, pp. 2499–2521, 2003.

[87] A. Mehta, J. Dürango, J. Tordsson, and E. Elmroth, "Online spike detection in cloud workloads," in *Proceedings of the IEEE International Conference on Cloud Engineering*, Mar 2015, pp. 446–451.

[88] J. Miranda, "How Etsy deploys more than 50 times a day," Mar. 2014. [Online]. Available: http://www.infoq.com/news/2014/03/etsy-deploy-50-times-a-day

[89] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*. Wiley, 2012.

[90] A. Morrow, E. Baseman, and S. Blanchard, "Ranking anomalous high performance computing sensor data using unsupervised clustering," in *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec 2016, pp. 629–632.

[91] M. A. Munawar, "Adaptive monitoring of complex software systems using management metrics," PhDThesis, Electrical and Computer Engineering, 2009.

[92] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, May 2010, pp. 114–117.

[93] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *Proceedings of the 20th International Symposium on Software Reliability Engineering*, Nov 2009, pp. 41–50.

[94] T. H. Nguyen, J. Luo, and H. W. Njogu, "An efficient approach to reduce alerts generated by multiple IDS products," *International Journal of Network Management*, vol. 24, no. 3, pp. 153–180, 2014.

[95] A. J. Oliner, A. Ganapathi, and W. Xu, "Advances and Challenges in Log Analysis," *Communications of the ACM*, vol. 55, no. 2, pp. 55–61, 2012.

[96] A. J. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007, pp. 76–86.

[97] ——, "What supercomputers say: A study of five system logs," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007, pp. 76–86.

[98] J. P. Onyango and A. Plews, *A textbook of basic statistics.* East African Publishers, 1987.

[99] J. W. Osborne, "Prediction in multiple regression," *Practical Assessment, Research and Evaluation (PARE)*, vol. 7, no. 2, pp. 1–9, 2000.

[100] H. S. Pannu, J. Liu, and S. Fu, "A self-evolving anomaly detection framework for developing highly dependable utility clouds," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, Dec 2012, pp. 1605–1610.

[101] J. Parekh, G. Jung, G. Swint, C. Pu, and A. Sahai, "Issues in bottleneck detection in multi-tier enterprise applications," in *Proceedings of the 14th IEEE International Workshop on Quality of Service*, Jun 2006, pp. 302–303.

[102] A. Patcha and J. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[103] ——, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[104] A. Pecchia and S. Russo, "Detection of software failures through event logs: An experimental study," in *Proceedings of the 23rd International Symposium on Software Reliability Engineering*, Nov 2012, pp. 31–40.

[105] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining*, Mar 2007, pp. 504–515.

[106] Ponemon, "Breaking down the cost implications of a data center outage," Ponemon Institute, Tech. Rep., Dec. 2013. [Online]. Available: http://www.emersonnetworkpower.com/en-US/Solutions/infographics/Pages/Cost_Implications_of_Outages.aspx

[107] R. Powers, M. Goldszmidt, and I. Cohen, "Short term performance forecasting in enterprise systems," in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 2005, pp. 801–807.

[108] E. T. Roush, "Cluster rolling upgrade using multiple version support," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, Oct 2001, p. 63.

[109] A. Sadighian, J. M. Fernandez, A. Lemay, and 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)*, Oct 2013, pp. 161–177.

[110] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and OANDA," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 21–30.

[111] M. Sladescu, A. Fekete, K. Lee, and A. Liu, "Geap: A generic approach to predicting workload bursts for web hosted events," in *Proceedings of the 15th International Conference Web Information Systems Engineering (WISE)*. Springer International Publishing, Oct 2014, pp. 319–335.

[112] J. Sondow, "Asgard: Web-based cloud management and deployment," 2012. [Online]. Available: https://medium.com/netflix-techblog

[113] C. Spence, L. Parra, and P. Sajda, "Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model," in *Proceedings of the IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA)*, 2001, pp. 3–10.

[114] J. Stearley, "Towards informatic analysis of syslogs," in *Proceedings of the IEEE International Conference on Cluster Computing*, Sep 2004, pp. 309–318.

[115] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *Proceedings of the 32Nd IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society, 2012, pp. 285–294.

[116] Z. Usmani and S. Singh, "A survey of virtual machine placement techniques in a cloud data center," *Procedia Computer Science*, vol. 78, pp. 491 – 498, 2016.

[117] R. Vaarandi, "SEC - a lightweight event correlation tool," in *IEEE Workshop on IP Operations and Management*, 2002, pp. 111–115.

[118] ——, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM)*, Oct 2003, pp. 119–126.

[119] R. Vaarandi, M. Kont, and M. Pihelgas, "Event log analysis with the logcluster tool," in *Proceedings of the IEEE Military Communications Conference, (MILCOM)*, Nov 2016, pp. 982–987.

[120] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM transactions on computer systems*, vol. 21, no. 2, pp. 164–206, 2003.

[121] C. W. VanVoorhis and B. L. Morgan, "Understanding power and rules of

thumb for determining sample sizes," *Tutorials in Quantitative Methods for Psychology*, vol. 3, no. 2, pp. 43–50, 2007.

[122] J. Varia, *Architecting for the Cloud: Best Practices*, 2011.

[123] Veeam Team, "Veeam data centre availability report," Veeam Software, Tech. Rep., Dec. 2014. [Online]. Available: http://go.veeam.com/2014-availability-report.html

[124] P. M. B. Vitanyi and M. Li, "Minimum description length induction, bayesianism, and kolmogorov complexity," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 446–464, Mar 2000.

[125] A. Wagner and B. Plattner, "Entropy based worm and anomaly detection in fast ip networks," in *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*. IEEE Computer Society, 2005, pp. 172–177.

[126] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online detection of utility cloud anomalies using metric distributions," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*, Apr 2010, pp. 96–103.

[127] T. Wang, J. Wei, W. Zhang, H. Zhong, and T. Huang, "Workload-aware anomaly detection for web applications," *Journal of Systems and Software*, vol. 89, pp. 19–32, Mar 2014.

[128] T. Wang, W. Zhang, C. Ye, J. Wei, H. Zhong, and T. Huang, "FD4C: automatic fault diagnosis framework for web applications in cloud computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 1, pp. 61–75, 2016.

[129] I. Weber, M. Farshchi, J. Mendling, and J.-G. Schneider, "Mining processes with multi-instantiation," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1231–1237.

[130] I. Weber, C. Li, L. Bass, X. Xu, and 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)*, Jun 2015, pp. 537–544.

[131] I. Weber, A. Rogge-Solti, C. Li, and J. Mendling, "Ccaas: Online conformance checking as a service." in *Proceedings of the International Conference on Business Process Management, (BPM)*, 2015, pp. 45–49.

[132] A. Wolski and K. Laiho, "Rolling upgrades for continuous services," in *Proceedings of the First International Service Availability Symposium (ISAS)*, May 2004, pp. 175–189.

[133] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. NA, 2017.

[134] W. Xu, "System problem detection by mining console logs," PhDThesis, University of California, Berkely, 2010.

[135] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. ACM, 2009, pp. 117–132.

[136] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Mining console logs for large-scale system problem detection." *SysML*, vol. 8, pp. 1–16, 2008.

[137] X. Xu, I. Weber, L. Bass, L. Zhu, H. Wada, and 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)*, Dec 2013, pp. 5:1–5:6.

[138] X. Xu, L. Zhu, M. Fu, D. Sun, A. B. Tran, P. Rimba, S. Dwarakanathan, and 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*, 23May 2015, pp. 69–75.

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

[140] X. Xu, L. Zhu, I. Weber, L. Bass, and 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))*, jun 2014, pp. 252–263.

[141] L. Yang, J. M. Schopf, C. L. Dumitrescu, and I. Foster, "Statistical data reduction for efficient application performance monitoring," in *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, vol. 1, May 2006, pp. 8 pp.–334.

[142] L. Yu and Z. Lan, "A scalable, non-parametric anomaly detection framework for hadoop," in *Proceedings of the ACM Cloud and Autonomic Computing Conference*. ACM, 2013, pp. 22:1–22:2.

[143] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. Jain, and 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)*, Oct 2014, pp. 249–265.

[144] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems Networks*, Jun 2009, pp. 572–577.

[145] Y. Zhu, N. M. Nayak, and A. K. Roy-Chowdhury, "Context-aware activity recognition and anomaly detection in video," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 91–101, Feb 2013.