Anomaly-aware Management of Cloud Computing Resources

Sara Kardani Moghaddam

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

July 2019

ORCID: 0000-0002-4967-5960

Anomaly-aware Management of Cloud Computing Resources

Sara Kardani Moghaddam
Co-Supervisors: Prof. Ramamohanarao Kotagiri and Prof. Rajkumar Buyya

---

Abstract

Cloud computing supports on-demand provisioning of resources in a virtualized, shared environment. Although virtualization and elasticity characteristics of cloud resources make this paradigm feasible, however, without efficient management of resources, the cloud system's performance can degrade substantially. Efficient management of resources is required due to the inherent dynamics of cloud environment such as workload changes or hardware and software functionality such as hardware failures and software bugs. In order to meet the performance expectations of users, a comprehensive understanding of the performance dynamics and proper management actions is required. With the advent of data analysis techniques, this goal can be achieved by analyzing large volumes of monitored data for discovering abnormalities in the performance data.

This thesis focuses on the anomaly aware resource scaling mechanisms which utilize anomaly detection techniques and resource scaling mechanism in the cloud to improve the performance of the system in terms of the quality of service and utilization of resources. It demonstrates how anomaly detection techniques can help to identify abnormalities in the behaviour of the system and trigger relevant resource reconfiguration actions to reduce the performance degradations in the application. The thesis advances the state-of-the-art in this field by making following contributions:

1. A taxonomy and comprehensive survey on performance analysis frameworks in the context of cloud resource management.

2. An Isolation-based anomaly detection module to identify performance anomalies in web based applications considering cloud dynamics.

3. An Isolation based iterative feature refinement to remove unrelated and noisy features to reduce the complexity of anomaly detection process in high-dimensional data.

4. A joint anomaly aware resource scaling mechanism for cloud hosted application. The approach tries to identify both the anomaly event and the root cause of the problem and trigger proper vertical and horizontal scaling actions to avoid or reduce performance degradations.

5. An adaptive Deep Reinforcement Learning (DRL) based scaling framework which leverages the knowledge of anomaly detection module to decide on proper decision making epochs. The scaling actions are encoded in DRL action space and the knowledge of actions values are obtained by training multi-layer Neural Networks.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Sara Kardani Moghaddam, July 2019

Preface

This thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-6 and are based on the following publications:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, Performance-Aware Management of Cloud Resources: A Taxonomy and Future Directions, *ACM Computing Surveys*, Volume 52, No. 4, Aug 2019.

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, Performance Anomaly Detection Using Isolation-Trees in Heterogeneous Workloads of Web Applications in Computing Clouds, *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 31, No. 20, ISSN: 1532-0626, Wiley Press, New York, USA, Oct 2019.

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ITL: An Isolation-Tree based Learning of Features for Anomaly Detection in Networked Systems, *Future Generation Computer Systems (FGCS)*(under 2nd review).

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ACAS: An Anomaly-based Cause Aware Auto-Scaling Framework for Clouds, *Journal of Parallel and Distributed Computing (JPDC)*, Volume 126, Pages: 107-120, ISSN: 0743-7315, Elsevier Press, Amsterdam, The Netherlands, April 2019.

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ADRL: A Hybrid Anomaly-aware Deep Reinforcement Learning-based Resource Scaling

in Clouds, *IEEE Transactions on Parallel and Distributed Systems (TPDS)*(under revision).

Acknowledgements

I would like to thank my supervisors, Professor Rao Kotagiri and Professor Rajkumar Buyya, for giving me the opportunity to undertake this PhD. I am truly grateful for their invaluable support, guidance and motivation throughout my candidature.

I would like to express my gratitude to my PhD committee, Professor Frank Vetere, for his comments and guidance during my candidature. I am also thankful to Dr. Rodrigo Calheiro for his constructive comments and technical advices in the beginning of my PhD journey. Special thanks to Dr. Sareh Fotuhi Piraghaj for support and valuable assistance on developing my research skills and improving my work. I would also like to thank all the past and current members of the CLOUDS Laboratory, at the University of Melbourne. In particular, I thank Dr. Adel Nadjaran Toosi, Dr. Maria Rodriguez, Dr. Amir Vahid Dastjerdi, Dr. Yaser Mansouri, Dr. Chenhao Qu, Dr. Bowen Zhou, Dr. Jungmin Jay Son, Dr. Safiollah Heidari, Dr. Liu Xunyun, Dr. Minxian Xu, Dr. Sukhpal Singh Gill, Caesar Wu, Farzad Khodadadi, Yali Zhao, Shashikant Ilager, Muhammad Hilman, Redowan Mahmud, Muhammed Tawfiqul Islam, TianZhang He, Mohammad Goudarzi, Zhiheng Zhong, Mohammad Reza Razian, Prof. Vlado Stankovski, Dr. Artur Pilimon, Dr. Arash Shaghaghi, Samodha Pallewatta, Amanda Jayanett for their support during my PhD journey.

I would like to express my sincerest thanks to all my friends in Australia and Iran, especially to my friend Suzan Maleki for her sincere friendship and supports which made living far from family much easier and happier.

I acknowledge the University of Melbourne and Australian Federal Government for providing me with scholarships to pursue my doctoral studies.

Finally, my deepest gratitude goes to my father, mother and brothers for their continuous support, love and encouragement at all times.

Sara Kardani Moghaddam
Melbourne, Australia
July 2019

Contents

List of Figures

List of Tables

Chapter 1
Introduction

With the advent of cloud era, the outsourcing of storage and computing resources as well as large scale computing-intensive applications are becoming more popular. Cloud allows the delivery of off-premise resided services where the complexities of hardware and software maintenance are transferred to cloud providers. The technology is impacting many organizations and industries in an optimistic manner. Nowadays, many individuals are exploiting the storage capabilities of cloud-based services such as DropBox and Google Drive; Many organizations are using the power of the cloud-based communication services such as emails and social networks; Many legacy systems are migrated toward the cloud to access more powerful resources with higher scalability and reliability. Moreover, the up-front investment in hardware, software implementation and maintenance or license costs can be avoided by utilizing fully deployed infrastructure and variety of services offered by cloud providers. According to *RightScale survey statistics*, in the year 2019, around 94% of respondents are using at least one public or private cloud with 79% of their workloads running on the cloud (Fig. 1.1)[1].

However, the flexibility of IT infrastructure in cloud systems brings a new era of challenges and opportunities in terms of the management of resources. Efficient management of computing resources is necessary to guarantee Service Level Agreements (SLA) in terms of the quality of delivered services. Small scale applications with a few numbers of clustered computing resources can be handled easily as both demands and supplies are predictable and controllable. In cloud computing, however, the resources are shared, the workload is heterogeneous and mostly unpredictable and the scale of management is large and distributed, comprising geographically scattered data centers with hundreds and thousands of physical resources. While over-previsioning of

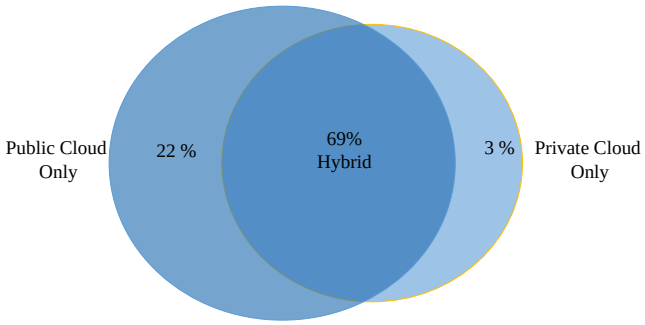


Figure 1.1: RightScale 2019 report on cloud usage statistics.

resources seems an easy solution for this problem, in reality, the resources are finite and wasted resources increase the costs and energy consumption. *RightScale2019* reports an estimated amount around 27%-35% of wasted resources in the cloud. The report also mentions that while the costs from wasted resources is a top challenge for cloud users, only a minority of companies are implementing automated policies to manage resources such as rightsizing instances [1]. Therefore, automated management of cloud resources to adapt to the real requirements of the environment is still a big challenge to be investigated. In this regard, the big decision for cloud providers is how to control the amount of resources to ensure the Quality of Services (QoS) as expected by the users while avoiding under-utilized states with wasted resources.

With the advances in the storage capabilities, a huge volume of log data from monitoring application and system level attributes has been provided for the administrators. These data provide a valuable source of traceable information on the performance of the system components. However, with this volume of data, manual policies are difficult to be enforced and tracked down. On the other hand, the advance in data analyzing and self-learning techniques is offering missing parts for an automated performance aware resource management solution for cloud providers. The idea is that the violations of QoS or the wastage of resources are detectable from the logs of the performance indicators, utilization metrics and other collected attributes from the environment. Therefore, by analyzing the recorded data, the system can provide information for questions such as when a problem happens, where and in some cases why it is happening and as a result trigger a proper response in terms of the allocated resources.

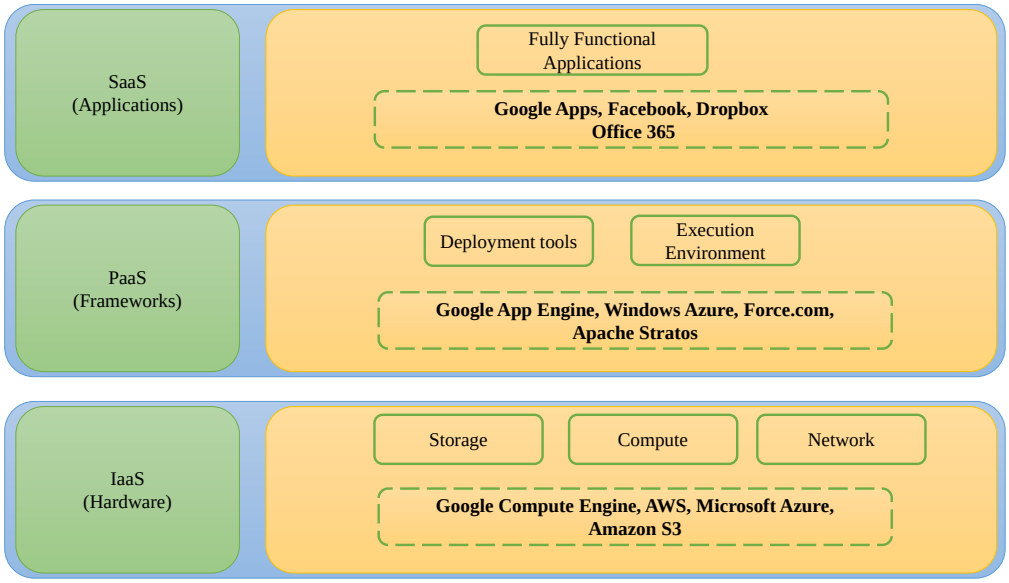


Figure 1.2: An abstract view of main models of cloud services.

This thesis addresses the problem of efficient management of cloud resources in the presence of performance problems using data analysis and anomaly detection techniques. Anomaly detection is used for identifying performance problems during the execution of cloud-hosted applications. We propose a detailed survey and taxonomy on performance aware resource management in cloud including various performance analysis techniques and corresponding resource adjustment solutions. Additionally, the applicability of anomaly detection is studied in terms of the effectiveness in cloud performance analysis and also high dimensional data. Then, a joint anomaly analysis and resource management module is proposed which demonstrates the efficacy of performance analysis in improving the quality of decisions in cloud resource management process.

1.1 Background

This section briefly reviews some of the main concepts and terms for this thesis.

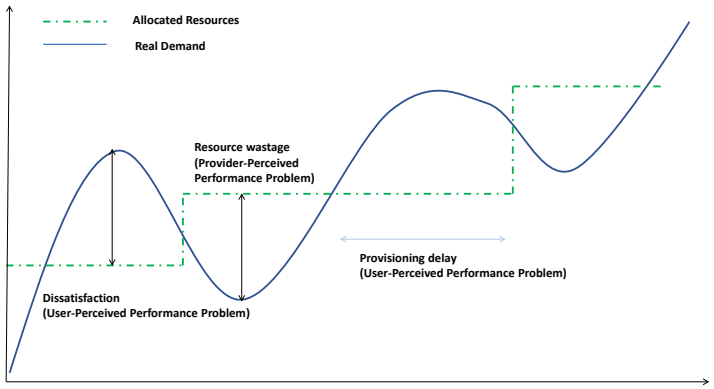


Figure 1.3: A simple view of resource provisioning performance problems.

1.1.1 Cloud Computing

Cloud Computing is a paradigm to offer a pool of resources (processing, storage, network, software, etc) over the Internet on an on-demand, pay-as-you-go policy. The offered resources can be categorized into three main models [2, 3] as shown in Fig. 1.2 :

- Infrastructure as a Service (IaaS): A model which provides the base resources such as computing, storage and network to enable customers to deploy and run their standalone applications, and operating systems. Examples of public IaaS providers include Amazon EC2 [4] and Google compute engine [5] which offer their resources with a variety of configurations and pricing models.
- Platform as a Service (PaaS): A model which offers the base platform including programming language tools, libraries and services to enable application developers to build and deploy their applications in the cloud.
- Software as a Service (SaaS): A model which provides applications and softwares such as email services, entertainment applications and social networks to be accessed by customers from interfaces such as web browser. Users can easily access the software through designed interfaces without worrying about installation or maintenance tasks.

IaaS is the lowest layer where the main interaction with hardware components is

happening. While physical resources are finite at this layer, virtualization and elasticity features of cloud create a virtual view of a "pool of infinite resources" for the upper layers. Virtualization enables the sharing of physical resources by creating the concept of virtual machines (VM). VMs are units of computing which can share the resources on the same Physical Machine (PM) while isolating the running of their own applications from the rest of the system. Elasticity is another feature of cloud which helps to add and remove resources automatically to adapt to the current demands in the system.

In this thesis, we focus on the problems of IaaS layer in terms of the elasticity of resources with regard to VMs. With the inherent dynamicity of cloud, originated from dynamic changes in the workloads, finite sharable resources and possible complexities in the functionality of software and hardware, making proper and on-time decisions about the configuration of resources is becoming more complicated. Static solutions can not capture the dynamicity of the environment and suffer the delayed decisions, violations of QoS or wasted resources as shown in Fig. 1.3. Therefore, resource providers have to continuously monitor the *performance* of resources and applications to make proper *decisions* at the right times to correct *performance problems* and maintain the expected Quality of Services(QoS). In the following, we briefly discuss the main terms in this definition.

1.1.2 Cloud Computing Performance

System Performance is usually defined in terms of the duration of performing a task or collection of tasks delivered as a service to the end users. For example, response time (RT) is a common attribute in web based applications that measure the duration from receiving a user's request to returning a response of the server. The concept of the performance is closely related to the definition of SLAs. From user's perspective, there is a range of variations in the performance that is acceptable and violating these levels requires the resource provider's attention or it causes penalties such as monetary costs. The expected services can be formulated in SLA contracts and service and resource providers are responsible for the fulfillment of these expectations. Therefore, the providers need to regularly monitor the demands and quality of delivered services to ensure the compliance of the SLAs.

There are two main approaches to measuring the performance of an application: 1. Performance is evaluated by directly targeting the metrics that evaluate the goodness of the functionality of services in the context of a specific application (ex. monitoring RT in a web-based service). However, this approach requires access to the VMs to evaluate the quality of the delivered services 2. Performance is evaluated indirectly by measuring the attributes of the running system. The idea is that if the running system shows signs of the problem in terms of the performance of resources (ex. CPU and memory utilization), these problems eventually affect the execution of hosted services. This approach may not be able to capture the application level problems, but it can be an effective way for detecting system-level problems especially when the decision makers do not have direct access to the target application and services.

1.1.3 Performance Anomalies

Performance problems are unexpected changes in the performance that may affect the experience of users with regard to the quality of delivered services. As they are abnormal compared to the normal behavior of the system, these problems are also known as anomalies. They are identified by continuously monitoring and analyzing the performance metrics and comparing the measurements with past behavior or known patterns of acceptable behavior. Quick identification of performance anomalies is a necessity to guarantee the satisfaction of SLAs. To better understand how these problems can impact the production environment, a real world problem is discussed in the following example:

In October 2011, Bank of America Online Banking suffered a series of slowdowns and outages in their site for six consecutive days which affected 29 million on-line customers. Bank attributed this problem to a combination of technical issues and higher than anticipated website traffic. The problem was noted as the result of a "multi-year project" upgrade. Testing of certain features and high traffic at the end of the month also contributed to the delays [6].

In the above example, the source of the problem is attributed to multiple reasons. However, the signs of the problem were identified with the experienced slow-downs of the service by customers. An autonomic resource management system should be able

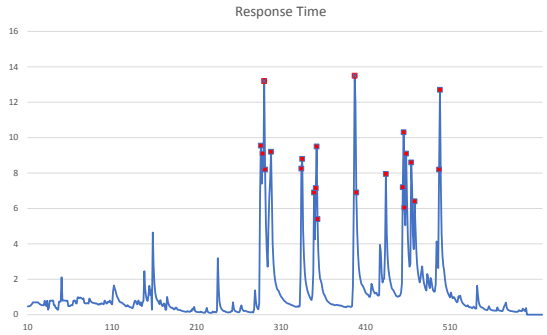


Figure 1.4: An example of latency anomalies in a plot of application response time.

to learn and predict the patterns of the traffic to pro-actively prepare extra resources for the duration of high load intervals without delays. Moreover, unexpected performance problems such as slowdowns and outages should be detected by monitoring the performance indicators and managed by reactive *reconfiguration* of resources. Fig. 1.4 is a plot of RT against time. The solid red dots are some examples of latency spikes that are detected as anomaly compared to the rest of the values.

Performance problems can happen at different levels of granularity such as hardware fault, software bug and mis-configurations, network attacks, and etc. In this thesis, we target a category of problems known as resource bottlenecks [7] which happens when the limitations of one or more resources cause performance degradation in the whole system. This is especially important for large scale cloud-based web applications where the stream of requests from users can change frequently. Unexpected changes affect the patterns of resource utilization with regard to the future requirements. For example, web applications are known to be prone to many performance problems which involve CPU and memory resources [8].

1.1.4 Resource Configuration Decisions

Resource configuration encompasses a variety of decisions including amount, type and location of allocated resources to an application. VMs are common unit of resource which can be configured. Public providers such as Amazon offer a variety of VM tem-

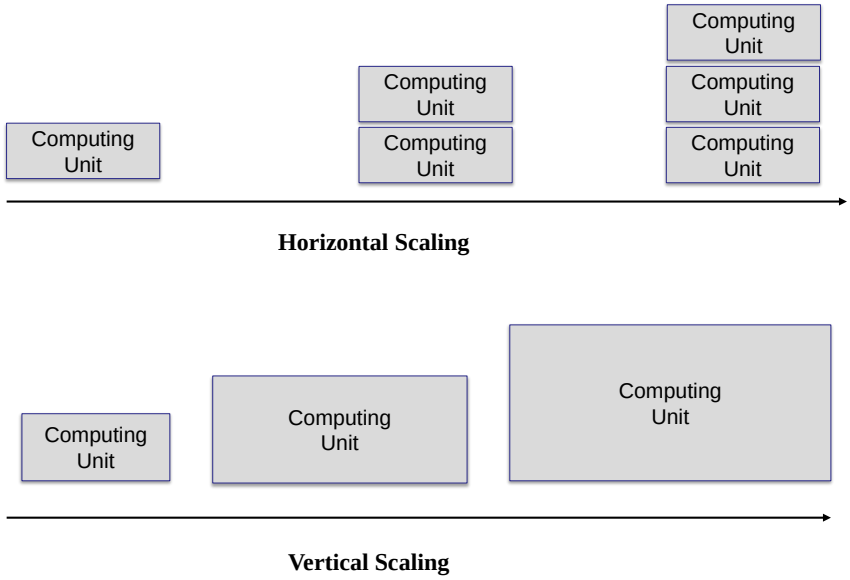


Figure 1.5: Horizontal vs Vertical scaling.

plates in terms of the CPU, RAM or storage to be added/removed (known as horizontal scaling solutions) during execution of the application. On the other hand, recent hardware and software advancements make fine-grained controls on resources (vertical scaling technology) possible. This enables the on-the-fly resizing of an active VM to adjust the amount of allocated resources to the workload of the VM. Horizontal and vertical resource configurations, as shown in Fig. 1.5, are two types of the scaling techniques that change the amount of active resources in the system. This thesis exploits both techniques as possible solutions for reactive and pro-active management of resources in response to the performance problems.

1.2 Research Problem and Objectives

This thesis addresses the adaptive scaling of cloud resources by exploiting performance anomaly detection techniques. The objective is to decrease the adversary effects of performance degradation in terms of the QoS and SLA satisfaction during execution of an application. Therefore, we formulate the problem as proposing an ***adaptive*** model for the ***scaling*** of resources in the presence of ***performance problems*** with regard to the fol-

lowing research questions:

- Q1: When a scaling decision should be triggered: To answer this question, we first need to answer when a performance problem happens in the system. A variety of techniques such as time series analysis, statistical profiling, machine learning and etc can be utilized to identify abnormal and unexpected patterns in the collected performance data. The final solution should be sound in terms of the accuracy of detected anomalies and quick to be applicable for real-time applications such as web-based systems.

- Q2: What type of the scaling should be triggered: Current literature mostly apply one of the above-mentioned scaling techniques with the majority of horizontal scaling. However, depending on the level of performance problems, one technique may outperform the other. For example, a system-level load problem (over-utilization) may require new VMs to be added. On the other hand, a VM-level resource shortage such as temporal increases of memory consumption from a background process can presumably be alleviated by a quick increase of allocated memory for target VM (Vertical scaling). The final solution should be scalable to fit the large scale applications in the cloud environment.

- Q3: How the model should be adapted to the changes: Adaptability of the model is a challenge to be tackled at both data analysis part and scaling decision maker. Anomaly detection models require regular updates to capture the recent normal state of the performance. Otherwise, the old models result in many false alarms and trigger unnecessary resource adjustments. On the other hand, the scaling decision maker requires a model to map the performance problem to correct scaling action. Considering the dynamics of the cloud environment, the decision makers may need to update their mapping rules to adapt to the new performance patterns.

1.3 Thesis Contributions

Based on the aforementioned research questions, the contributions of this thesis are as follows:

1. A comprehensive taxonomy and survey of performance aware resource management in cloud, including performance analysis solutions and corresponding resource adjustment actions.

2. Investigating efficient anomaly detection in the context of the cloud performance and high dimensional data (Q1)

   - Deploying an anomaly injection module on a web-based testbed to generate web performance data with a variety of performance anomalies.
   - A time-series based model for identifying performance problems in web performance data
   - Validation of anomaly detection accuracy with a multi metric based approach with regard to both AUC and PRAUC
   - A new Bagging based anomaly detection technique
   - A novel iterative feature refinement to make the anomaly detection testing faster and feasible for high-dimensional data.

3. A novel distributed joint anomaly triggering and rule-based computing resource management framework (Q2)

   - A cause inference model to identify the cause of the performance problem.
   - A model updating algorithm for on-line updating of anomaly detection models.
   - A two-level distributed algorithm for combining horizontal scaling (System level) and vertical scaling (Local VMs) actions.

4. An adaptive Deep Reinforcement Learning (DRL) based resource management (Q3)

   - Exploiting RL learning concept to improve the self-adaptivity of resource management in the context of real-time applications with performance problems.

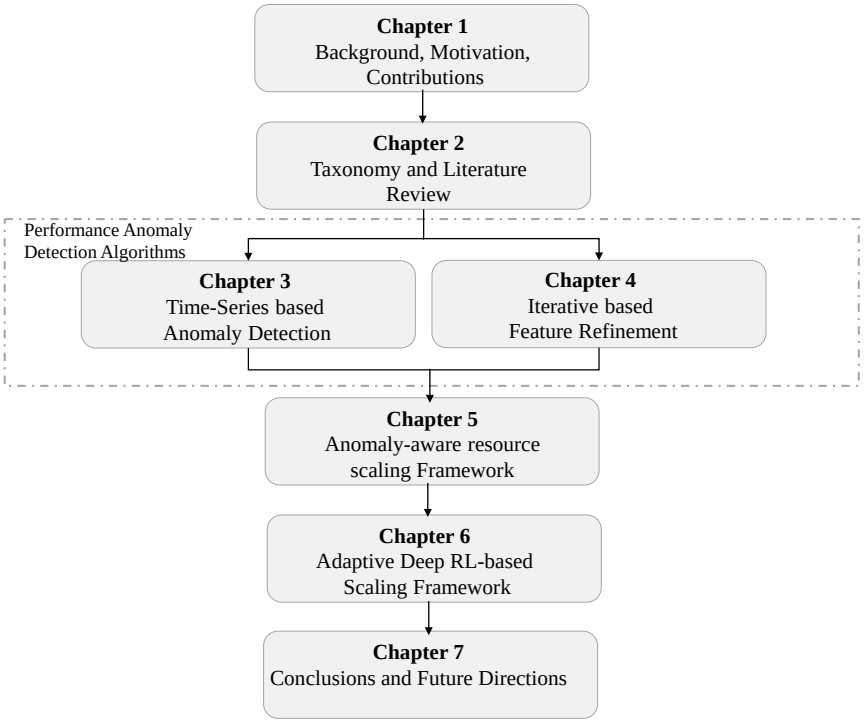


Figure 1.6: The thesis structure.

- A new event-based RL framework for scaling decisions. Decision epochs are defined based on the anomalous events instead of conventional time-based epochs.
- Combining scaling solutions into RL action space to adjust the amount of VM-level CPU and memory as well as the number of VMs in the system.
-

1.4 Thesis Organization

The structure of this thesis is shown in Fig. 1.6 and Fig. 1.7 and is derived from several conference and journal papers published during PhD candidature. The remainder of this thesis is organized as follows:

- Chapter 2 presents a taxonomy and survey on anomaly and workload aware resource management in the cloud. This chapter is derived from:

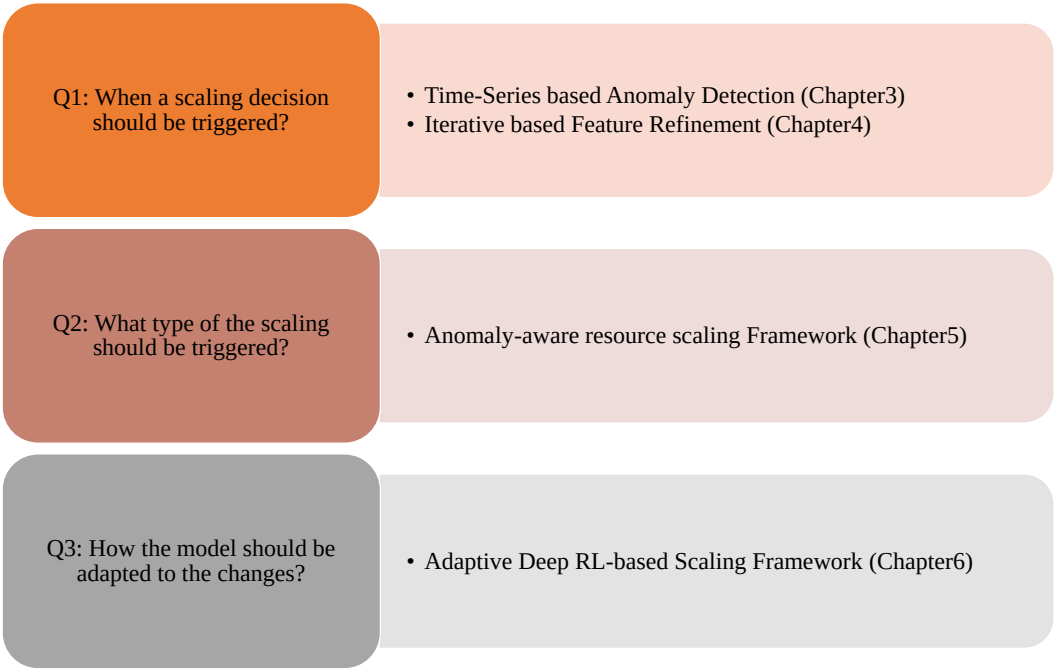


Figure 1.7: The connection of research questions and thesis chapters.

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, Performance-Aware Management of Cloud Resources: A Taxonomy and Future Directions, ACM Computing Surveys, Volume 52, No. 4, Aug 2019.

- Chapter 3 presents a fast time-series based anomaly detection approach for dynamic and heterogeneous workloads of web applications in cloud. This chapter is derived from:

  - **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, Performance Anomaly Detection Using Isolation-Trees in Heterogeneous Workloads of Web Applications in Computing Clouds, Concurrency and Computation: Practice and Experience (CCPE), Volume 31, No. 20, ISSN: 1532-0626, Wiley Press, New York, USA, Oct 2019.

- Chapter 4 presents a novel iterative algorithm to refine features of high-dimensional data with the aim of improving the efficiency of anomaly detection in terms of the training and testing times. This chapter is derived from:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ITL: An Isolation-Tree based Learning of Features for Anomaly Detection in Networked Systems, *Future Generation Computer Systems (FGCS)*(under 2nd review).

- Chapter 5 proposes a novel anomaly-aware computing resource management solution with a two-level combination of horizontal and vertical scaling actions for SLA fulfillment. This chapter is derived from:

  - **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ACAS: An Anomaly-based Cause Aware Auto-Scaling Framework for Clouds", Journal of Parallel and Distributed Computing (JPDC), Volume 126, Pages: 107-120, ISSN: 0743-7315, Elsevier Press, Amsterdam, The Netherlands, April 2019.

- Chapter 6 proposes an adaptive Deep-Reinforcement Learning based resource management with anomaly-based decision epochs. This chapter is derived from:

  - **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ADRL: A Hybrid Anomaly-aware Deep Reinforcement Learning-based Resource Scaling in Clouds, *IEEE Transactions on Parallel and Distributed Systems (TPDS)* (under revision).

- Chapter 7 concludes the thesis by summarizing the findings and discussion on future works.

Chapter 2

A Taxonomy and Review of Performance-aware Management of Cloud Resources

This chapter proposes a taxonomy to depict the main approaches in existing researches from the data analysis side to resource adjustment techniques. The main requirements and limitations in resource management including a study of the approaches in workload and anomaly analysis in the context of the performance management in the cloud are discussed. The detailed survey of existing approaches and their classification based on the proposed taxonomy is presented. Finally, considering the observed gaps in the general direction of the surveyed works, a list of the new research gaps for future researchers is proposed.

2.1 Introduction

Cloud computing as an on-demand, pay-as-you-go environment has been modelled based on two main concepts of elasticity and virtualization. The inherent flexibility brought by these techniques in the area of high performance computing is accompanied with the complexity of managing distributed resources while meeting the expectations of the users. The emergence of the public Cloud Service Providers (CSP) such as Amazon and Google which are extending the scientific limited applications of the cloud environment to industrial, academic and personal use cases make the need for more advanced

This chapter is derived from:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, Performance-Aware Management of Cloud Resources: A Taxonomy and Future Directions, ACM Computing Surveys, Volume 52, No. 4, Aug 2019.

and complex resource management solutions highly important.

The main goal for CSPs is to find better ways of utilizing resources while keeping the service level agreements (SLA) as expected. SLAs are contracts among CSPs and customers to maintain the minimum Quality of Service (QoS) delivered by the offered applications. Breach of the SLAs costs the CSPs both money and their reputation. Considering dynamic characteristics of cloud including unreliability and heterogeneity in resources and workloads, simple static resource planning solutions do not work. Therefore, traditional resource management architecture is extended with monitoring modules which can provide timely information on the performance of the application along with the resource utilization of system components. The collected data from monitoring the system and application provide a source of highly valuable information about the health of the system. On the other hand, advances in data learning methods have provided missing parts of a data aware performance management offering all the concepts and tools for analyzing data to find patterns, trends and interesting changes in the behaviour of monitored components. The integration of two parts of performance data analytics and automated resource management brings new challenges and opportunities in both areas of theoretical concepts and practical implementations. In this chapter, we try to identify the major challenges and corresponding solutions in the problem of data aware performance analysis and resource management in the cloud. We present a taxonomy to depict various perspectives of the performance management in the cloud, covering all aspects of data collection, analytics, and resource adjustment solutions.

The rest of this chapter is organized as follows: Section 2.2 describes the main blocks of performance data aware resource management and existing challenges followed by listing the most influencing factors in this area. Section 2.3 introduces two main approaches in utilizing data as a source of extra knowledge for resource management. Then, sections 2.4 to 2.8 review different characteristics of data analysis and resource management modules based on the categories identified in the taxonomy. Finally, section 2.9 discusses the main gaps and directions for future researchers and section 2.10 summarizes the chapter.

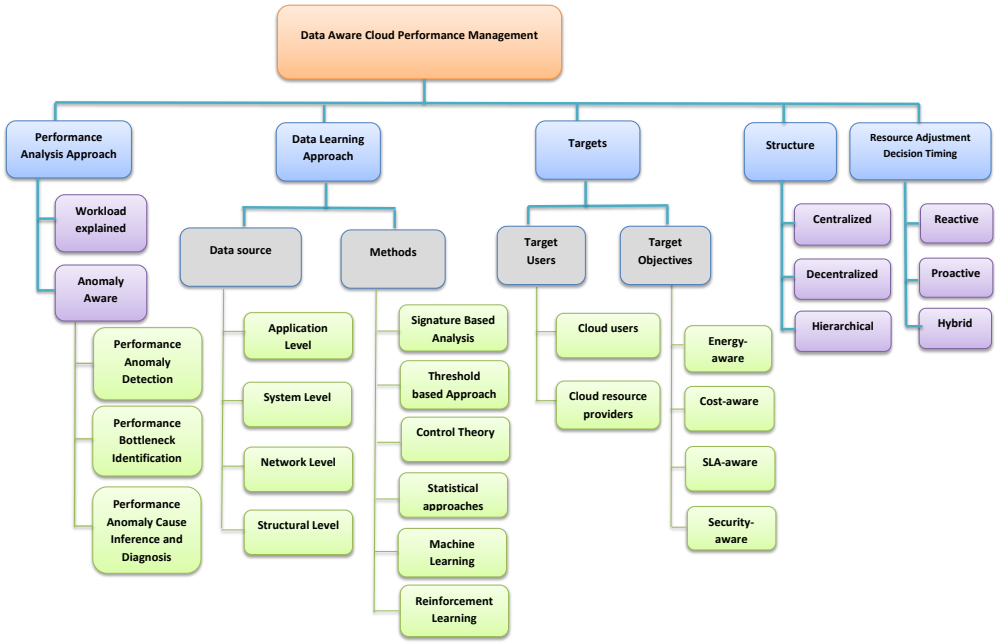


Figure 2.1: The Taxonomy of Performance Data-aware Management of Cloud Computing Resources.

2.2 Background

The concept of resource management in the cloud environment encompasses all the techniques and procedures that help to adjust the configuration of resources according to the demands of the users and applications in the system. For example, auto-scaling solutions are based on a characteristic that allows system resources to expand or shrink automatically at different levels of granularity (Virtual Machines (VMs), CPU, RAM, etc) according to the perceived state of the system. To be clear about these concepts, we pursue the following definitions in the rest of the chapter:

Performance Indicators: All the measurable attributes from the resources and applications which demonstrate the degree of functionality of the corresponding unit in the system. These indicators continuously change over time and are initial sources of information for the health of the system. For example, the time it takes to load a page known as *response time* (RT) from a web based application is the most perceptible sign that the system is performing at the expected level or not. Longer than usual RTs trigger the warnings of having some sort of the problem, requiring technical considerations form

the system administrators.

System State: State or behaviour of the system at each time is an abstract representation of all the operational attributes and performance indicators of the system which can be recognized in normal or abnormal/anomalous condition.

The main indicators of an abnormal state are the presence of the unexpected patterns or values in the performance indicators of the system.

Performance Degradations are caused by abnormal behaviors when they affect the performance indicators adversary. For example, in the case of the increase in the number of requests (increased demand from customers) to a web server, if current resources cannot handle the newly received requests, the RT observed by users will increase. The unacceptable increases in the RT are considered as performance degradation which should be avoided. One solution can be to add new resources corresponding to the overloaded component of the application so the amount of resources is in accordance with the incoming load to the system.

Considering aforementioned definitions, any automated Resource Management Module (RMM) is dealing with two main challenges:

When a performance degradation is happening in the system? In an ideal, highly reliable environment where no abnormal behaviour is expected and applications show consistent behaviour with stable performance, traditional static scheduling solutions will work and dynamic scaling of the resources is not required. However, in a real environment with a wide range of internal and external factors which can affect the behaviour of the system, performance degradation has become an important challenge to be dealt with accurately. There is a wide range of causes identified for these problems from fluctuations in the incoming workload to a malfunctioned hardware or a buggy software that can affect the performance of the application or VMs. Therefore, the onset time of the degradations should be known so a proper and timely corrective action can be started. Monitoring sensors which track the performance of each component generate a vast amount of data which include hidden patterns and signs of the health of the system. Previously, we had to rely on the expert of the human operators to skim the data and find alerting behavior. However, considering the scale of the generated data from hundreds and thousands of machines located in different geographical locations, the manual

approach is not feasible anymore. Therefore, researchers have started to take advantage of the advanced data analytics methods and more powerful and cost-effective computing hardware to automate and accelerate the process and find better quality knowledge about the performance of the target systems.

What type of corrective action should be performed? In order to alleviate the performance problems of the system, RMM should start a corrective action in the form of load redistribution, resource provisioning, migrations, etc. Current resource providers such as Amazon or Azure offer migrations or simple threshold based scaling services which change the number of VMs in the system. There are also more customized resource management policies such as on the fly changes in the resource configuration of one VM which is offered by some CSPs such as [9]. The selection of proper action can be dependent on many factors including technical or business limitations, type of the problem, etc. We have identified some of the most important factors as follows:

- **Technical limitations:** Virtualization is the key concept for cloud models. It enables hosting different applications or the components of one application independently on one Physical Machine (PM) with migration capability available to move them to other PMs without significant downtimes in the system. Currently, many public resource providers such as Amazon and Microsoft Azure offer the required environment for CSPs to host their applications on VMs and dynamically add/remove VMs in the system. There are also more fine grained controls available to configure resources at VM-level known as vertical scaling. In this process, size of the VM can change on-the-fly without any rebooting of the VMs. However, the functionality needs support from both hypervisor and the kernel of the VM. Currently, providers such as Amazon [4, 10] and Microsoft Azure do not support this functionality.

- **Business considerations:** There are a vast amount of the resources offered by cloud resource providers with various pricing strategies. For example, Amazon offers on-demand instances with hour/seconds based pricing or much cheaper reserved instances with long-term contracts [4]. There are different pricing rules for vertical scaling of the VMs such as offered rules by [9]. CSPs should consider these

options when deciding on the configuration of their system and scaling policies. As a result, selecting the best action will be limited to the available budget predefined by the application owners. For example, in the case of the budget shortage, some levels of the performance degradations may be acceptable from the owner's perspective.

- **Root cause of the problem:** In traditional threshold based scaling, changes in the number of the VMs is the most common response to performance problems in the system. However, there is a wide variety of reasons from hardware faults to local software bugs in the application or security issues such as Distributed Denial of Service (DDoS) attacks that can create the signs of performance degradations. In cases that resource shortage is not the main reason for the problem, adding new instances to the system may temporally alleviate the problem, but it is not optimal as a long-run solution. Moreover, as the vertical scaling is becoming more prominent as a scaling option, having the knowledge of the underlying reason has become more interesting for a more cost or resource effective solutions. For example, in the case of a local memory shortage in one VM, a VM-level increase of the available memory may be more effective than adding new VMs. A more detailed explanation of the pro and cons of these types of decisions are presented in section 2.8.

- **SLA agreements**: SLA agreements are contracts between users and CSPs which identify the expected QoS received by customers. These expectations are usually based on the outputs of the system perceivable by customers such as the availability of the service or the delays in the response. Having specific requirements for the output of the system may limit available choices of the RMM. For example, CSPs may consider over-provisioning as a better option than dynamic scaling to manage high loads in the system when having a stable response time is highly important for the customers.

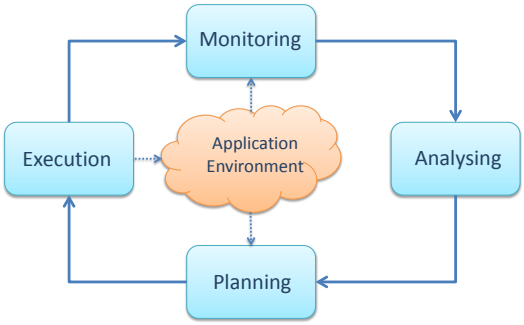


Figure 2.2: General Phases of Data Aware Performance Manager in Cloud

2.2.1 Data Aware Resource Management

Motivated by the aforementioned challenges and requirements, researchers are leveraging various tools and concepts to offer more mature solutions for cloud resource management. An area which has been vastly investigated is data analytic techniques which are bringing new opportunities and challenges in the area of distributed performance management. In order to apply these techniques, researchers are focusing on the obtainable knowledge from the data collected from performance indicators of the system and applications. It has been shown that these data are a valuable source of information on the health of the system and a starting point for detecting initial symptoms of the abnormal behaviors. Based on the selected performance data to be monitored, the approach for the abstraction and modeling of the system and the actions that are performed to mitigate the performance problems, different types of the resource management strategies are proposed. In order to better understand the building blocks of these solutions, we first briefly review four main components of data aware performance management framework in the following paragraphs.

The main parts of the automated resource management in the cloud can be explained based on a classic control loop known as MAPE (Monitor, Analyse, Plan, Execute) loop [11] which is shown in Figure 2.2. The performance of the system is continuously monitored and a range of attributes from resources and applications are collected. The collected data are cleaned, modelled and analyzed to identify any symptom of changes in the normal behaviour of the system. Finally, based on the output of the analysis phase, a proper action is selected and the target components are informed to start the execu-

tion of the action. We briefly explain each phase and list the related categories of the taxonomy to each part as follows:

Monitoring: The performance of the system can be tracked by collecting the values of the attributes from the components of the system. These attributes include all the workload metrics, system traces, network features or performance indicators of the system such as CPU and memory utilization, the number of incoming requests, the number of threads or response time of the application. The **Data Learning Approach** of the taxonomy indicates different levels of the data collected during the monitoring phase.

There are a variety of tools to help monitoring and collecting data from system components including *Top* and *Iostat* packages or *Ganglia* framework [12]. One can select a proper tool based on the factors such as the granularity of data to be collected, the level of access to the system components, scalability and characteristics of the system.

One point worth mentioning is how to select a proper monitoring interval time. The interval can be selected as small as 1 second or a large value as 1 hour. Smaller intervals make it possible to capture the fast changing patterns or fluctuations with higher accuracy. However, the amount of the storage required for keeping all the recorded data and the overhead of processing and cleaning of the data significantly increases. Selecting larger intervals reduces the overhead and required storage, but the possibility of missing or delayed detection of changes in the pattern of the performance data increases which can cause delayed triggering of the corrective actions and more SLA violations. One should select a proper interval considering the trade-off between the accuracy and computation complexity, the reliability of the environment and type of the application [13, 14]. For example, one approach is to define the sampling interval as a function of the dynamicity of application by following the pattern of changes in the workload or performance indicators and adjusting the sampling interval accordingly. Another approach follows a fine-grained dynamicity analysis which considers the behavior of metrics separately. In this approach, the monitoring intervals can be tuned at the metric level by having larger intervals for the metrics with no change points in the past data. Smaller intervals are selected for highly dynamic metrics which their changing behavior is directly impacting the performance indicators of the application.

Analyzing: In general, two main blocks of the data analyzer module are data prepa-

ration and performance modeling/analyzing. Therefore, all the steps required for cleaning and filtering, dimensionality reduction, building the models, analyzing new observations and deciding on the model updates when the state of the system changes are parts of this phase. A wide range of techniques and algorithms can be used to learn a model based on the historical behaviour of the system. The **Data Learning Approach** and **Performance Analysis Approach** parts of the taxonomy present different categorization of existing methods for this phase.

Planning: The inputs for the planning module are the information about the current or future state of the system from the analyzer, the current configuration of the resources from the application environment and the objectives and constraints from customers or resource providers. Depending on the obtained knowledge, the module can select from a range of possible actions such as adding/removing VMs, changing the configuration/placement of multiple VMs or inbound traffic balancing. Decisions can be formulated based on past experiments and knowledge about possible causes of changes in the system. Therefore, the process can be implemented as a simple sequence of If-else rules or at a larger scale, as a database that can map a combination of influential parameters to their corresponding mitigation action. The subcategory presented in Figure 2.5 focuses on the this phase. A detailed explanation of possible actions can be found in section 2.8.

Execution: This is where the final execution of planned actions in the system is performed. The module utilizes existing libraries and APIs to communicate with application components or deployed VMs to add new resources, remove the idle ones, change the configurations of existing VMs or updating load balancer configuration files. This phase more concerns the development strategies and techniques which are out of the scope of the current research.

In the following sections, we present different aspects of a data aware resource management solution based on the categories shown in the Figure 2.1 and subcategories presented in 2.4 and 2.5. Based on the categories and identified approaches, we map each work to the corresponding features in Table 2.2 to give the readers a quick view of the main contributions of each work.

2.3 Performance Management in Cloud

Monitoring tools collect a valuable source of the data to be analyzed and provide a timely update on the performance state of the application and resources. Data learning approaches offer the researchers all the necessary concepts and tools to sift through the collected data and predict the future behaviour or find interesting patterns of unexpected behaviors or anomalies with their possible causes. In this section, two main approaches for analyzing the performance of the system are presented.

2.3.1 Workload-driven Performance Management

Performance of the system can be modelled and predicted based on the workload-related features such as the number of requests received or the amount of processing required at each time interval. Di et al.[15] propose a method for long-term load prediction in Google data centres. They consider load in the system as the main factor affecting the performance of the system and ignore other sources of data. In order to have a better representation of the statistical properties of the load including trends and seasonality, different metrics based on the load measurement values are derived. The prediction is done by training a Bayes classifier and exploiting a time window approach which is a suitable way to smooth high fluctuations in the load. However, other types of anomalies which can be directly related to specific performance metrics can not be detected by this approach meaning that unexpected behaviour can occur in the system, possibly causing negative impacts on the user experience. Work presented by Cetinski and Juric[16] considers a single attribute, number of required processors at a certain time, to estimate the utilization of resources. They expand the training dataset by introducing new attributes based on similar patterns in historical data. The results show that these new attributes improve the prediction accuracy of Random Forest compared to K-Nearest Neighbor algorithm. However, their prediction does not include the concept of unexpected behaviors resulting from various anomaly sources. VScaler proposed by Yazdanov and Fetzer[17] leverages a combination of workload prediction and reinforcement learning (RL) to automatically scale VM resources with regard to the user-provided SLAs. RL approach in this framework helps to automate the learning process, considering the

uncertainty of environment in the form of changes in the workload model of the application. Another work by Yang et al.[18] presents a cost aware resource auto-scaling mechanism which considers both costs of adding new VMs as well as business software license during scaling up procedure. A combination of linear regression based workload prediction, integer programming based pre-scaling and threshold based real-time scaling is introduced for capacity planning and resource management. The real-time scaling can be considered as a reactive step to compensate prediction errors, but the simulation based validation of this approach ignores many complexities and time requirements of mentioned methods and these assumptions must be carefully verified.

In the workload explained performance management approach, the changes in the pattern of the workload are the primary influential factor that can affect the performance and hence the resource decision making of the system. The definition of the workload is dependent on the application and can be demonstrated as the number of requests sent to an interactive application such as web based systems, number of tasks/jobs running in the system and etc. One can also consider the resource demands of the jobs to be processed at each time as a representation of the existing load of the system. However, this assumption should be verified whether the resource consumption is a sole function of the load of the target application or the dynamic factors such as the effect of background applications and sharing of resources are considered in the process.

2.3.2 Anomaly Aware Performance Management

A different approach to address the problem of performance management targets the abnormality in the system behaviour as a starting sign for possible problems to be addressed by the resource management module. In the context of big data enhanced solutions, these performance problems are considered as outliers and anomalies in the data which can be identified by utilizing a variety of methods such as statistical or machine learning algorithms. Therefore, we first define the term of anomaly in a general context; Then, the problem of identifying anomalies in the context of the cloud is explained and existing works that follow this approach are discussed in more detail.

What are Anomalies?

Anomalies are the patterns in the data that do not conform to the usual behaviour of observed data. The concept of anomalies and anomaly detection area are very general and presented under different names including outliers and novelty detection, finding surprising patterns in data, fault or abnormal behaviour detection in the systems [19]. These areas have been investigated over a long period of time as part of the medical and clinical data clustering, image processing and surveillance cameras, financial fraud detection, and several other applications.

Performance Anomaly Identification in Cloud Environment

In general definition of anomaly detection, the goal is to model the normal behaviour of the system, so any unexpected change in the patterns can be seen as an anomaly. However, considering the user centric approach in resource management decisions in the cloud, the application owners are more interested in the events that can affect the performance of the system and degrade the quality of service experienced by the user. We refer to all of these events as performance anomalies. Considering that performance degradations can cause resource wastage, loss of reputation and cost penalties for cloud service and resource providers, many researchers have investigated the relation between measurements from the system and application-dependent performance indicators to have a better understanding of different causes of performance problems. We identify three levels of knowledge obtained from the process of performance anomaly analysis as shown in Figure 2.3 which are detailed in the following paragraphs:

Performance Anomaly Detection The goal of a data analyzer module at the performance anomaly detection level is to find any abnormal pattern in the behaviour of the system that can be a symptom of the performance problems. Therefore, the input for these frameworks is usually the system and application performance indicators while the output is a performance alert when an anomaly is detected in the system. Having this goal and considering the fact that a correlation of different metrics can be related to various types of anomalies, Guan and Fu[20] present an automatic anomaly identification technique for adaptive detection performance anomalies such as disk and memory

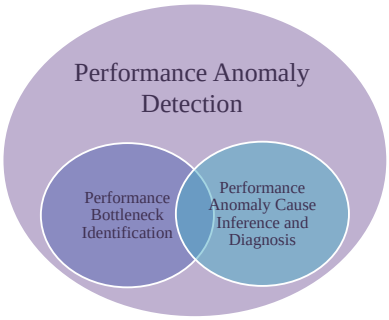


Figure 2.3: Different levels of knowledge from performance anomaly analysis

related failures. Proposed method investigates the idea that a subset of principal components of metrics can be highly correlated to specific failures in the system. A combination of Neural Network method and Adaptive Kalman Filter is utilized in a procedure of learning from historical data, updating the prediction models based on the current prediction errors and adapting to the newly detected anomalies to improve the detection performance. The work presented in [21] focuses on two general categories of anomaly sources, workload related and performance related data in streaming servers. They justify this separation as a requirement to select the best repair action in the response to degradations caused by targeted faults. A feature selection procedure based on Naive Bayes is employed and the most relevant features are reported. Ashfaq et al.[22] target the problem of anomaly detection from a new perspective, highlighting the scalability problem of data analytic solutions for resource management issues in the cloud environment. They propose a general framework for anomaly detection based on the splitting feature space into multiple disjoint subspaces and applying anomaly detection methods on each subspace separately. The idea behind using feature space slicing is to decrease the likelihood that a high number of normal instances can average out the effect of a few dispersed numbers of malicious instances during anomaly identification. Since this approach requires higher computation resources, as it needs to run multiple simultaneous instances of the algorithm on different subspace, it is more suited for high performance computing platforms.

BARCA (Behaviour Identification Architecture)[23] is a framework for online identification of anomalies in distributed applications. It divides the anomaly detection pro-

cess into two steps. First, a one-class classifier distinguishes normal behaviour from unexpected ones. Then, a multi-class classifier is used to separate different types of abnormal behaviours. The framework generates time series of different collected performance data and extracts new features such as skewness and mean of data which better represent the characteristics of the time series and also help to reduce the dimensionality of feature space.

The above-mentioned works target the reactive anomaly detection problem in the cloud environment. In order to be able to move the system back from abnormal to the normal state with minimum negative impact, we need to know about the probability of having abnormal values in the future. In the proactive approaches, systems are able to exhibit goal-directed behaviour by anticipating possible future abnormalities and taking initiatives [24]. Gu and Wang[25] investigate proactive anomaly detection in data stream processing systems. Their proposed solution includes a phase of predicting resource utilization and then applying an anomaly identification algorithm on predicted data. Considering time sensitiveness of stream data, proposed procedure is online and the classifier will be updated periodically based on the new data. To address the prediction problem, they apply Markov chain to capture changing pattern of different metrics to predict future resource utilization. Markov chains are based on the idea that future state only depends on the current state and not the past values. This assumption can be problematic, especially for the data with recurrent patterns and events. Tan et al.[26] address this problem by integrating a 2-dependent Markov model as the predictor with Tree-Augmented Naive (TAN) Bayesian networks for anomaly detection. Another study by [27] investigates unsupervised behaviour learning problem for proactive anomaly detection. The proposed framework uses Self-Organizing Maps (SOM) to map a high dimensional input space (performance metrics) to a lower dimensional map without losing the structural information of original instances.

Performance Bottleneck Identification Performance bottleneck identification goes one level deeper in the process of finding anomaly events in the data, trying to find possible bottleneck metrics that are closely related to the observed performance degradations as well. This approach is closely related to the problem of resource management as it targets finding possible system resources that need to undergo a reconfiguration so

the provided resources meet the requirements of the application. Tan et al.[26] leverage TAN to distinguish normal state from abnormal ones as well as reporting the most related metrics to each type of the anomaly. Canonical correlation analysis and Support Vector Machine (SVM) based feature selection are used by FD4C framework[28] to diagnose faults in the web applications. They utilize a recursive approach based on the feature elimination to rank the most important metrics for each type of the anomaly. Xiong et al.[29] have a different approach for detecting the performance bottlenecks. They try to find the most relevant metrics to the performance of the application and follow the changes in these metrics as a sign of the performance problems. However, they show that the predicted metrics are also good indicators of the source of analyzed performance problems, pointing to the source host and type of the bottleneck resource. UBL presented in [27] uses the topological properties of SOM to compare the anomaly and normal states and identify the metrics that are different between these states as faulty metrics.

Performance Anomaly Cause Inference and Diagnosis The aforementioned anomaly detection approaches mostly focus on detection of the abnormal symptoms and a coarse-grained identification of the possible *resource level metrics* that contribute to the performance degradations. However, none of them dig deep into the data obtained from the application to find the underlying reasons for the observed problems. Indeed, identified bottleneck metrics can be the indicators of having an application or VM level fault or inconsistency in the system. For example, high incoming load to the application or a faulty loop in the software code which saturates the CPU of the VM or the problem of VM/application contentions which may cause degradations in memory utilization.

We can identify different directions in the fine-grained analysis of the source of the faults. First, the works that aim to localize the source of the fault to one *component* such as nodes, VMs or application components. For example, the works done in [30, 31] address the fault localization problem in distributed applications. The proposed frameworks combine the knowledge of inter-component dependencies with change point selection methods, taking into account that abnormal changes usually start from the source and propagates to other non-faulty parts based on the interactions of the components.

Another direction is to distinguish among different *types* of the faults. Dean et al.[32]

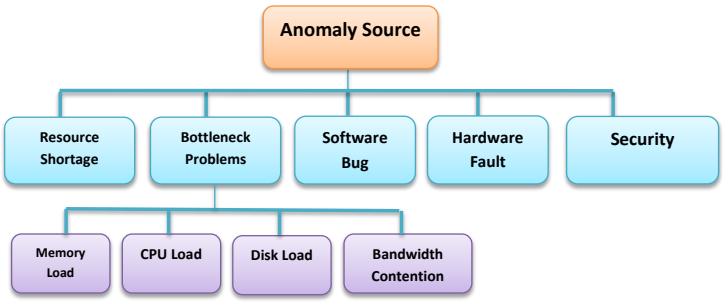


Figure 2.4: Source of Performance Problems

propose PerfCompass which analyzes the generated system calls in the system to distinguish between internal and external faults. They focus on software related bugs such as endless loops as the target internal faults. Cid-Fuentes et al.[23] apply a set of the SVM based binary classifiers to distinguish among livelock, deadlock and starvation faults.

To achieve a more fine-grained identification of the cause, Dean et al.[33] propose PerfScope to analyze the anomalies occurring due to the software bugs of the application. The framework studies the patterns in the *system calls* and tries to find anomalous interactions between user and kernel. Triage[34] is another online software failure diagnosis which identifies the conditions as well as the *code* and variables involved in the failure state. TaskInsight presented in [35] focus on the thread and process level performance information which helps to localize the problem to the target anomalous task.

Cloud Performance Anomaly Root Causes

Cloud application owners typically start to allocate resources based on the recommended application requirements and then change resource configurations by continuously monitoring performance indicators to find performance violations. The root cause of these performance problems can vary widely as shown in Figure 2.4.

Hardware faults include the problems that are originated from the corrupted or performance degraded hardware that host target applications [36, 37]. Software related problems can be caused by a buggy code in the application or misconfiguration that

causes inconsistency in the functionality of software or interactions among components. This type of problem also can be caused by network related bugs and misconfiguration that happen at the application level, including reported bugs in skype, MySQL or IPv6 compatibility issues [38].

Security problems including attacks are another source for unexpected behaviours caused by the unusual pattern of requests such as successful port-scans, attacks on the application server, etc. A wide variety of literature target this area, proposing various types of intrusion detection systems based on the concepts of statistical feature analyzing, classification and clustering [22, 39, 40]. In order to identify these types of the problems, one needs to collect network layer datasets including packet header information or the frequency of sender IP addresses to detect unusual patterns in the requests [22, 41]. Other sources of data to help recognize the access patterns to the application are server log files which record the history of authentication and user access requests over time.

Resource shortage issues are another reason for the performance problems when the lack of enough resources to satisfy existing requests causes service interruptions and degradations in the performance. The limitation can be due to the budget constraints or business policies that do not allow adding extra resources to the system or reduces the amount of the existing resources. Unexpected termination of instances performed by resource providers is one case of these problems that can expose the system to a state that the performance and throughput of the system will be degraded [42].

Bottleneck problems are another reason for the performance problems which are caused by insufficient resources in one or more components of the application. The problem can be due to the specific requirements of offered services such as working with a CPU intensive software and the lack of consistency among the application demands with provided resources. Another example is the effect of the background processes which can temporarily saturate the resources of the machine, ignoring the requirements of other installed applications. It is worth noting that if the components are dependent and have interactions, the bottleneck problem in one part of the system can quickly affect other dependent components and as a result, the performance problem will propagate in the system causing application performance degradations [25].

2.4 Target

The body of literature in performance aware resource management addresses the effectiveness of triggered actions from different perspectives in terms of the target area of the final solution. We distinguish two main factors that can affect this decision and accordingly design of the proposed frameworks. These factors are explained in the following paragraphs.

2.4.1 Users

To establish a distributed, shared, pay-as-you-go environment, different players with heterogeneous and even conflicting objectives should be able to cooperate. Current literature mostly recognizes 3 main players/layers, cloud resource provider (CRP), cloud service provider (CSP) and final users as contributing roles in the design of the resource management frameworks [43]. CRP, also known as the infrastructure provider, provides access to the pool of resources in the form of physical machines, storage, network and other types of resources which are necessary blocks for creating a distributed computing environment. This can happen as a direct access to the physical resources or through virtualization technologies, usually as the units of virtual machines. CSPs perform as the interface between CRPs and the final users and offer a range of services hosted on computing resources leased from CRPs. The final users are the customers that demand the cloud hosted services by sending requests and data through predesignated interfaces. The separation of the layers is not always clear and sometimes more than one role can be performed by one entity. For example, some CRPs also offer customized software packages as a service without the intervention of third-party providers [5]. Another approach distinguishes a broker layer which acts as a mediator between CRPs and CSPs. The broker has the information from both interactors including user SLA and resource prices and usually performs negotiations with multiple resource providers to find the offer which best meets the SLA requirements [44]. Although these layers have each their own responsibilities, from our perspective in the study of performance management, we identify two main groups as the target users as follows:

- *Cloud users/Application owners* who have access to the application dependent infor-

mation including the code-level data, components design, workload patterns and QoS requirements at the VM level.

- *Resource providers/owners* who have information on the hardware characteristics of PMs and make decisions on VM allocation/placements.

Depending on the target users, resource management can be adapted to address the improvements of measurable metrics in the favor one or both groups [45, 46]. The selection of the target users can affect the selection of the objective as well as the source of the data to be processed in the learning procedure. The objective of RMM determines the direction of decision makings in terms of the measurable metrics to be improved. Moreover, each group of the users has access to different sources of data and can help the RMM decision making through managing various parts of application, network and hardware configurations. We have explained these concepts in sections 2.4.2 and 2.6, respectively. It is worth noting that in all cases, while one user group may be mentioned as the main executor of the proposed solution, other groups can also be involved when the final decisions indirectly affect their respective goal attainment. For example, CSPs may need to be considered when the final decisions (such as VM migrations/scalings) can violate the budget constraints or predefined security concerns.

2.4.2 Objectives

Depending on the beneficiary of the proposed solutions, a variety of metrics are selected as the objective of the improvements during the resource management decision making. As it is shown in Figure 2.1, we identify 4 main categories with regard to the target objective metrics. It is worth noting that a combination of these objectives is usually considered in a trade-off based optimization problem or through a list of the constraints provided by users.

- *Energy:* Energy-aware solutions try to minimize the power consumption of system through a variety of mechanisms including application optimizations, dynamic scaling, VM configuration, allocation and consolidation techniques [47]. Energy consumption of system is usually calculated with regards to the resource utilization of PMs. Consolidation techniques concern the minimization of the active PMs

by placing VMs so that the machines can be utilized effectively while reducing the number of underutilized machines as much as possible [48, 49]. VM placement [48, 50] and VM reconfiguration including hardware tuning techniques [51] are other solutions that try to save energy by improving the performance of machines and reducing the power usage of hardware components. Load distribution is also among traditional approaches of SLA aware resource management which can also be used for the purpose of energy saving by prioritizing low-energy or renewable powered datacenters during the resource allocation process [52]. These solutions are mainly focused on resource provider objectives and require direct access to the resources and information that may not be available for the service providers. In contrast, solutions that address the energy problem from service providers perspective focus on the component level optimization of applications and green software designs [53, 54]. For example, [53] proposes an algorithm to dynamically select application components to be deactivated as a response to the performance degradations in an overloaded machine. This helps the cloud service to be responsive to the users during the high loads on the system by keeping the non-essential components of the applications in suspended mode.

- *Cost:* Considering the model of pay-as-go as the basis of cloud systems, market based models focus on the monetary value of offered resources and services for cloud providers and consumers [45, 55, 56]. These models provide solutions to optimize the total cost of executing tasks considering a variety of pricing models. Resource utilization metrics such as memory and CPU consumptions, storage, bandwidth and etc are among measurable metrics to be attributed as the cost indicators in proposed solutions. The cost effectiveness of the final solution usually inversely impacts the QoS values (higher delays or runtime). Therefore, it is a common approach to consider a trade-off between the cost and target performance metrics as the objective function to reduce or penalize the adverse impacts on the QoS [55, 57].

- *SLA:* Alternatively, a large body of literature addresses the problem of resource management with more customized approaches to target the specific application

requirements. These requirements are usually noted in the SLA contracts and their violations incur penalties. A variety of indicators such as response time, service up-time and scalability are included in this category [45, 58, 59]. For example, [45] addresses the availability aware resource provisioning by considering service agreements on the minimum accepted up-times. The violation of these SLAs causes penalties which are taken into account in the optimization problems.

- *Security:* Security aware solutions offer mechanisms to protect the safety and integrity of users, data, applications and underlying infrastructure [60]. Alongside intrusion aware frameworks which make use of the traffic or VM related data to detect possible attacks on the application, this category also deals with policies to protect data including user data provided to the application or user behavior patterns. A variety of solutions are combined to secure the integrity, availability and confidentially of data during phases of data life cycle in the system [60]. These solutions can be coarse grained approaches including VM migration/placements and workload isolation or fine-grained solutions that directly target the data security by encryption, isolation and cleansing techniques [61, 62].

2.5 Architecture

In an environment with geographically distributed resources, the decisions on the placing and interaction among components can highly affect the performance of the system and corresponding corrective actions. Various factors such as the amount of available storage, resource demands or the speed of information dissemination can contribute to these decisions. In the following, we briefly discuss three main approaches for the placement of different components of a resource management framework in the environment.

2.5.1 Centralized

Traditional frameworks to analyze the health state of the system, typically follow a centralized approach. In a centralized structure, all data from local components including performance indicators and resource configurations are sent to a master node. The mas-

ter node is responsible to maintain a continuously updated model of the whole system and triggers alarms when a performance problem is happening. All the tasks about workload prediction, resource utilization estimation or performance problem analysis are done at this module [63–65].

An advantage of a master node approach is to have all system related performance information in one place; therefore, one can analyze the interactions and relation among the metrics at different layers and tracks the fault propagation among connected components. However, as the scale of the system increases, there will be more components and resources to be monitored usually in geographically distributed regions. The process generates a huge volume of collected data to be transferred and analyzed in one place. This makes the system modeling and abstraction as well as triggering a proper resource management action to be traversed across all the involved resources, super complex, computationally intensive and time consuming.

2.5.2 Distributed

In a dynamic and scalable system, administrators are more inclined to deploy computing modules as decentralized components [25, 26, 66]. Therefore, each VM/physical machine or small clusters of machines in the system will have a local analyzer dedicated to processing locally collected monitored data to model the behaviour of the system and locally decide the resource configurations. This approach can be easily deployed and has higher scalability and manageable computation time. Moreover, failures in one node do not affect the functionality of remaining processing modules. This approach is also used in Fog-computing based Internet Of Things (IoT) infrastructures to model a fine grained connectivity that extensively uses the advantages of having multiple layers of distributed computing for a highly scalable environment to connect people and devices [67]. Distributed computing helps to significantly decrease the decision-making time as it deals with smaller environments (monitoring one VM or host compared to the whole environment) with reduced amount of data or problem-causing factors.

2.5.3 Hierarchical

While distributed architecture solves the scalability issue of the centralized approaches, the lack of a central manager makes it hard to include the interaction and dependency among components during analysis. Each module has an abstract performance model of its local environment, completely ignoring the effect of any external factors such as the dependency among different layers of multi-tier applications. Moreover, the distributed modules do not have a big picture of the system to decide on resource adjustments at higher levels of granularity such as adding new VMs. As a solution for this problem, one can consider a combination of the centralized and distributed architecture in a hierarchal model, where each monitored component has a local analyzer module to get up-to-date information on the local behavior and trigger local corrective actions [68]. They can also share a summary of the local state of the component to a central module [30, 69]. Central module utilizes a combination of the knowledge from local states, the dependency among components and high level information of the functionality of local components to create a general model of behaviour for the whole system. This approach combines the scalability and flexibility of distributed methods with system wide knowledge of the centralized master node which helps the system to respond to local problems quickly while having enough knowledge to diagnose and plan for the global problems.

2.6 Data Learning Approaches

Data learning module of the RMM receives a set of the raw data as input and tries to extract up-to-date information on the performance state of the system to be sent to the decision making module. Depending on the objective of the solution as described in section 2.4.2, the required metrics are collected from the system and processed with a variety of data analysis techniques including statistical and machine learning approaches. A discussion of the sources of data to be monitored and the techniques for analyzing data are presented in the following sections.

2.6.1 Data sources

Monitored metrics are the measurable features which provide a basis to describe and model the state of the system. Depending on the target users and the objective of the proposed solutions, a variety of metrics from system resources and application components can be collected. In general, we have identified four primary sources of data that give information on the system from different perspectives to be processed by the analyzer.

System-Level: System level metrics refer to the collection of attributes that can describe or predict the behaviour of running environment including VM or physical machine. One category of these attributes is resource level metrics which act as the performance indicators of the running system at different levels of granularity from VMs to specific processes and threads. For example, one can present the number of assigned CPU cores or the percentage of used CPU, Memory or Disk I/O at different time intervals as indicators of the functionality of the system during runtime of the applications. Another source of data which can be categorized as part of the system metrics are generated system calls which show the pattern of interactions with operating system services. It is shown that these patterns can be affected by different types of internal and external faults which help to detect and localize the source of the faults [32, 33].

Many cloud resource providers offer monitoring services to collect data from physical and virtualized machines. Amazon CloudWatch is an example of these services. There is also a range of system monitoring and application debugging tools such as htop, Iostat and strace, each offering a level of information about the utilization of resources or pattern of interactions among processes in the system. While these tools give valuable information about the functionality of one machine, their usage for a cluster of machines needs more scripting and data management. For a more flexible monitoring of the distributed systems, advanced frameworks such as Ganglia are introduced [12]. Ganglia is a distributed framework based on the hierarchical design which uses technologies such as XML and RRDtools for monitoring different components of the system. It is accompanied with a dashboard to view live statistics of the monitored system while the recorded data is also available for deeper analysis. A detailed analysis of characteristics of various monitoring tools is presented in [70].

One point worth mentioning here is that some of the above-mentioned tools require one to directly access the monitored VMs, or some components of the monitoring modules should be installed on the machines beforehand. Therefore, the outputs of these monitoring components are accessible by cloud application owners (*cloud users*) who have access to the VMs. Moreover, cloud users have direct access to the components of the application. Therefore, QoS aware solutions that require the knowledge of the performance of the system are designed with the assumption of having an access level same as the application owners. Alternatively, there are works that follow the black-box rules, trying to avoid or decrease the dependency on the application or guest VMs by utilizing hypervisor capabilities to collect data from outside of the VMs [16, 26, 65]. These solutions can be utilized by cloud resource providers. A combination of having VM-level information and knowledge accompanied with the capability to access the underlying hardware (including hardware tunings, VM/Storage server placement) gives the resource providers a great power to manage and adjust the utilization as well as SLA such as privacy and security requirements of applications owners [14].

Application-Level: Application level metrics are collected from application components deployed in the system. Since these metrics are directly related to the runtime state of the application, they can be very informative and good indicators of the health of the application or environment. For example, in a web based application, response time which is the delay from initiating the request until the user receives the results is considered as an indicator of the quality of the offered web service. Longer response times can be warning signs for a high number of requests or some problem inside the systems [18, 26]. There are also more fine-grained works that study the software code and debug the flow of data, compare the effect of various input variables or environment configurations [34]. These works are more related to the field of software debugging or runtime diagnosis. However, we have included them in our survey as they can help to distinguish internal application faults from external ones which consequently affects the type of corrective actions from service provider side.

Network-Level: There is a body of works which study the obtainable knowledge from analyzing network level data such as packet headers or the frequency of received packets from specific users which makes them suitable for identifying network related

issues and particularly security threats such as DDoS attacks [22, 41]. The source of the problem in these cases is usually associated with the external factors and therefore these frameworks are complementary to the ones that directly target the internal state of the system and resources.

Structural-Level: While a per component monitoring gives valuable information on the functionality of individual parts of the system, these components are in continuous interactions in a distributed system. In other words, the functionality of one part may be dependent on the correct execution of another part. Therefore, the fault in one component can quickly spread based on the application architecture and the path of flow of the data/commands among components. Having the information on the execution order of application components along with the timestamped data of performance metrics can give new insights into localizing actual sources of the faults which are propagated from different layers of the application [30, 31].

2.6.2 Methods

The core part of the data aware resource management is the data analysis module which obtains the knowledge on the current or future state of the system to select the best action and keep the system compliant with the SLA requirements. There are different approaches to help learning and analyzing the health of the system from collected measurements. We categorize and summarize the characteristics of the identified approaches in the following subsections. It is also a common approach to combine two or more of these techniques for different parts of data analysis and decision making modules.

Signature Based Analysis

A state in the system can be characterized by the values of the attributes of its components at different levels of granularity. Considering that various types of the faults or performance problems leave distinctive signs on the attributes, one can capture a snapshot of all values during abnormal/normal behaviour and represent it as the fingerprint of this state. The works presented in [71] and [72] distinguish performance problems

caused by anomalies from the ones which are the results of an application update or the changes in the resource consumption models. They create a profile of the application performance based on the concept of transaction processing times and corresponding resource utilization. The profiles are used for the comparison between old and new application performance to detect the changes. While these works leverage the application related measurements to create profiles, Brunnert and Krcmar[73] utilize resource profiles to detect the performance changes in the enterprise applications (EA). The resource profiles are defined as the amount of required resources including CPU and memory for each transaction of an EA version. Resource profiles are not dependent on hardware characteristics or workloads, therefore, they are more robust solutions for areas such as capacity planning or energy estimation. Another approach is to use the profiles of events for diagnosing the type of anomalies. For example, Sharma et al.[74] propose a fault management framework which first detects an anomalous behaviour by statistical analyzers. Then, detected deviations are matched with the pr-defined signatures of the faults to identify the cause of the problem.

While the signature based approaches usually show low false positive rates, an important challenge is the creation of baseline profiles which capture the target states of the system. Regarding anomaly detection problems, creating signatures need the domain knowledge of the problem and there is always a high chance of missing unknown anomalies which increases the false negative rates in the results.

Threshold based Approach

Threshold based approach is a simple yet popular way among cloud providers to define a set of rules to manage the resources in the system [4]. The idea behind this approach is that anomalies can cause an unusual increase or decrease in the utilization of the resources, affecting the values of attributes of the system or application. One can define the scaling up/down rules by identifying a threshold for the acceptable utilization in the system. If the target utilizations exceed the threshold, scaling actions are triggered. Therefore, two main parts of each threshold based rule are the condition and the action. Regarding the condition part of the rule, an attribute of the system which can be a re-

source level metric or application level performance indicator is selected. Then, proper lower/upper thresholds are identified. Whenever a threshold value is exceeded, the conditions are met and the action is started. The second part of the rule is defining appropriate actions such as deciding on the number of VMs to be added or the VMs which can be shut down in the system. These actions, also known as horizontal scaling policies, are offered by most of the cloud resource providers. Gmach et al.[75] investigate the reactive threshold based approach to detect over-utilized or underutilized servers. Yang et al.[18] extend this approach with a linear regression based prediction phase and apply one of the vertical or horizontal policies when a violation of the threshold is met. There are also works which implement the threshold based policies for the baseline comparison with their proposed frameworks [76, 77]. For example, Hong et al.[77] compare Markov based anomaly detection scheme with a threshold based monitoring which triggers anomaly alerts when the resource utilization thresholds are violated. While this approach is very common and easy in terms of the implementation, it requires a deep understanding of the workload patterns and trends. Considering the dynamic nature of today's applications, threshold based solutions can not fulfill the complex -sometimes conflicting expectations- of application and resource owners.

Control Theory

To improve the degree of adaptation to the changing environment, the mathematical concepts of control loops are investigated to create more responsive strategies for the dynamicity of the environment. Control loops help to automate the resource scaling decisions by creating a systematic way of adapting to the changes in the system. The controller should trigger proper corrective actions by adjusting the values of input variables to maintain the output or controlled variables close to a baseline. The process usually is designed as a loop with a variety of metrics from the system as input. The outputs are translated to some type of the action in terms of the system configuration adjustments. Regarding *Open-loop controllers*, the corrective action is selected solely based on the inputs, while in the *closed-loop* also known as *feedback controllers*, the changes in the controlled variable are received as the feedback to be considered by the controller

for the next action. While the latter is the most common architecture for implementing the adaptability in the dynamic environments, two other extensions of this architecture known as *Observe-Decide-Act (ODA)* and *Monitor-Analysis-Plan-Execute (MAPE)* are also exploited in the area of autonomic cloud resource management [78]. The ODA frameworks typically cover 3 main roles of observer, decider and executer with the main goal of decoupling the responsibility at different steps among respective players involved during system development process [79]. The separation of responsibilities allows application and system developers to deeply focus on the knowledge from their part of the problem and also makes the final system more adaptable to different applications and systems. The MAPE framework is another extension which breaks the action part of the general loop into two subproblems of analyze and execution. Following this architecture, Aslanpour et al.[80] propose a cost aware auto-scaling framework with the focus on possible improvements at the execution level. Nevertheless, in general, the feedback controllers are the most common architecture to be investigated in this section. Integration of the Kalman filter and feedback controllers are studied in [81] to manage the allocation of the resources based on the CPU utilization of VMs. Al-Shishtawy and Vlassov[82] combine feedback and feedforward controllers, harnessing the power of both approaches for multi-tier applications. Feedforward part is acting as a predictive controller to proactively avoid SLA violations caused by unexpected increases in the workload. In the case of the violations, feedback controller reacts to compensate the deviations in the performance.

Lyapunov control is another approach for solving optimization problems especially for online decision makings where the detailed information on the system behavior is not available. The Lyapunov based systems are also applied in optimization problems when the stability of the system (for example the queuing delays) is a point of interest [83, 84]. For example, Lu et al.[84] uses this approach to provide a cost minimization model with controllable provisioning delays in an auction based resource management. This approach has been shown to perform better compared to traditional methods in terms of dealing with the dimensionality problems in large systems without the preknowledge of statistical features of controlled components [85].

Proportional-Integral-Derivative (PID) based controllers are another technique ex-

ploited in [86] for managing the number of VMs in the system, aiming at keeping the service quality in accordance with the agreement levels. Alternatively, considering the need for more flexible systems with the capability of regulating more than one metric, Persico et al.[43] combine a fuzzy based scheduling algorithm with a PID controller for horizontally scaling of the resources. In contrast to the model based controllers which need some knowledge of the dynamics of the environment [81, 87], PID based controller makes the system more flexible in a model free adaption, usually through the iterative tunings of the parameters. However, despite the inherent simplicity of PID controllers to cover a broad range of applications, they may suffer from the oscillation or delayed convergence especially in highly unstable systems.

Predictive models are another approach that are commonly applied for complex highly dynamic environments. Model Predictive Controllers (MPC) consider dynamic models of the system over consecutive time slots (time-horizons). The current state as well as the future predictions are taken into account to repetitively optimize the controller model. APPLEware is a distributed MPC based middleware that optimizes both energy usage and the performance for co-located VMs [88]. The control actions are in the form of CPU and memory changes and are computed based on the predictions of energy and performance over prediction horizons to optimize the cost function at corresponding time intervals. Another instance of the predictive models is Receding Horizon Controllers (RHC) [89] which constantly solve optimization problems over a moving time horizon [90]. The iterative optimization of RHC is used in [91, 92] to minimize the cost for reserved and on-demand VMs while meeting constraints on the response times of application. Similarly, Roy et al.[93] apply this model for minimizing resource allocation costs considering both costs of leasing resources as well as the penalty costs of SLA violations. Incerto et al.[94] exploit the iterative optimization of RHC for fine grained adaptivity at the application level. The control model is used to automate parameter tuning of software components that are modeled based on queuing networks. While the iterative optimization of these models helps better adaptability to the changing environment, it also increases the runtime complexity for solving the optimization models at each step.

Statistical Approaches

Statistical approaches usually assume that the key attributes of the system follow a known or inferable behavior. Therefore, observing and collecting the data on the system attributes provides a baseline which any deviation from that is identified as an anomaly. The definition of the baseline behaviour is usually based on some statistical characteristics of the data such as mean and standard deviations. For example, many works on the anomaly detection are based on the assumption that values of the target attributes follow a normal distribution. Multivariate Adaptive Statistical Filtering (MASF) is a common method in this group which tries to find multiple sets of the control limits based on the statistical analysis of the previous measurements of the features during normal system operations [95]. The observations outside of the control limits are considered as possible anomalies in the system. Wang et al.[96] generalize this concept to more flexible thresholds, being more adaptable to dynamics of the workloads in data centers.

While these solutions are simple and lightweight, the highly dynamic nature of the cloud requires more flexible solutions which can capture the relation among features. A body of works [97–99] try to show some type of the correlation among resource metrics and QoS indicators, using this information for better understanding of the nature of the anomalies and further performance analysis such as cause identification. Another approach which addresses the problem of proactive resource scaling leverages regressions based methods to find the relation among metrics and performance indicators [18, 29, 65, 100]. The prediction of the future workloads or resource utilization gives insights on the possible changes in the system that require a reconfiguration of the resources. For example, MLscale is an auto-scaler which uses the regression method to predict the values of the metrics and consequently the performance indicators of the system through a hypothetical scaling and decides on the best action based on the results [100].

Another area within the domain of distributed resource management, where statistical techniques have been commonly used, is analyzing network level state of the system which helps to distinguish between normal traffic and network related security issues such as attacks. Gu et al.[101] employ relative entropy to compare the new traffic data with the baseline distribution and identify anomalous traffic. Cao et al.[102] target DoS

attacks launched by malicious tenants of the VMs in cloud data centers. Entropy is calculated based on the resource and network utilization information. They show that the entropy of VM's status drops when the attack starts. Ashfaq et al.[22] divide the feature space of the problem based on the information content concept, putting statistically similar instances in the same subspaces. The idea behind this approach is to avoid the effect of averaging out of anomaly points and also localize noise artifacts in separate subspaces.

Machine Learning

Machine learning concepts include the techniques that enable a system to learn from the experience over time without being explicitly programmed. The massive amount of the collectible data from the system is a valuable source of the information to be utilized by these techniques to learn from the environment. Each technique tries to structure data in a different way to generate an abstract model relating the input to output variables. Generally, we can divide these techniques into two main categories Supervised and Unsupervised. In the following, we explain each category in more detail.

- *Supervised Learning* Supervised algorithms require the dataset to be labelled, meaning that the desired output should also be clear during the training phase. This approach is more suitable for problems that the goal is to find a mapping between the input and output variables, so having the new input observation one can find the possible output. A common case of utilizing this technique is the classification of a set of records when each input should be assigned to one of the predefined classes. In the area of anomaly detection techniques, the classifiers can be used to categorize different types of anomalies or more generally distinguish between normal and anomaly state of the system. Following this approach, Gu and Wang[25] try to detect type of the future anomalies by using naive Bayesian classifier. A set of binary classifiers is trained to distinguish among different types of bottleneck problems. The authors show that the proposed classifiers can achieve high accuracy, detecting the anomaly symptoms caused by some of the common bottleneck issues at the application and resource level. Similarly, Tan et al.[26] exploit TAN to

predict the anomaly state of the system. Cunha and Moura e Silva[21] apply two classification algorithms, J48 Trees and Naive Bayes, on the historical data through ten-fold cross-validation. The goal is to differentiate between workload related anomalies that are caused by higher request rates and other types of performance anomalies. Decision trees are leveraged in [62] for classification of traffic data by supervised learning of attack types. The authors suggest that the simplicity and interpretability of trees can help humans to better understand the nature of problems and their related features. Supervised learning is also applicable for learning of the features. Accordingly, Shi et al.[103] exploit the knowledge from principle component structures and data labels to learn a more robust set of features for traffic classification.

Neural Networks are commonly applied for the prediction of the future utilization, performance indicators or workload metrics to model the performance of the application [65, 100, 104]. Rather than a direct identification of anomaly events (only considering the context of the data and patterns), these works usually focus on finding the symptoms of performance degradations in the application. In contrast, Guan and Fu[20] identify anomaly events by combining neural network and kalman filters to adaptively calculate the principal components of data. The idea behind their approach is that a subset of principal components is more related to specific types of failure in the system.

The aforementioned models are trained to find a relation between input and output variables to predict output values for test instances. The outputs can be related performance metrics such as response time or a category of anomaly states assigned to the input instance. A major limitation of these works is the requirement to have a labelled dataset for the training. The process of labelling a dataset is time consuming and needs a good knowledge of the domain problem. Moreover, in a dynamic environment, there is always a chance that the underlying mapping of the variables changes which require continuous reconfiguration and regeneration of the models for the new states of the system.

- *Unsupervised Learning* In contrast to the supervised learning, unsupervised ap-

proach sifts through data trying to find hidden structures and patterns. Therefore, it does not need any prior information about the labels of training data. The objective is to cluster input data based on their features without any assumption about their distribution [35]. The unsupervised learning is particularly suited for the cloud environment, where the system administrators may not have access to the detailed VM utilization states or be unaware of the internal performance of the application. Following this approach, Dean et al.[27] propose Unsupervised behaviour Learning (UBL) and investigate unsupervised learning problem for proactive anomaly detection. UBL is a framework which applies Self Organizing Map (SOM) to identify the anomalous states in cloud systems. SOM is an unsupervised type of the artificial neural networks that projects data instances from a high dimensional space to a lower space (usually two) while keeping the topological structure of data. Comparing the neurons of the generated map, they distinguish normal and anomaly states while a list of ranked metrics can also be inferred as a starting hint to find the cause of the problems. Hidden Markov Models (HMM) are used in [77] to model system as a Markov process. In a Markov process, it is assumed that the state of the system at each time is only dependent on the previous state. Two hidden states normal and anomaly are determined and the probability matrices are initialized through an unsupervised training process. A two-phase clustering approach is proposed in [105] which tries to find a VM placement solution to minimize the number of physical machines as well as reducing the performance degradations caused by the contention between co-located VMs. Hierarchical and K-means clusterings are applied to cluster VMs based on the peak of utilization metrics and the correlation among them. Alternatively, Ashfaq et al.[22] apply clustering as a preprocessing phase to divide feature instances into distinctive categories based on their statistical attributes. These clusters form the basic blocks of data to be analyzed separately by anomaly detection modules.

While the traditional methods of behavior learning can help to identify anomalous events, the detection is usually a by-product of other purposes such as clustering/classification. Isolation-based technique is another unsupervised approach that addresses this problem by directly targeting the characteristics of anomaly in-

stances which are few and different than normal instances [106, 107]. This method is leveraged in [108] to design a sequential unsupervised learning of the features where the calculated scores from Isolation-based trees are used as a signal for the selection of a subset of features for next iterations.

Unsupervised learning helps the system to detect both known and unknown anomalies. The process does not assume any prior knowledge about the statistical features or patterns in the data and tries to find common characteristics observed among different sets of the instances. However, depending on the level of the details provided in data, the accuracy of unsupervised learning to recognize the exact categories of anomalies may be affected.

Reinforcement Learning

Reinforcement learning (RL) focuses on the gradual learning through sequential interactions of the agents with the environment. The target goal of the agent is to maximize a reward function by selecting the best possible action based on the state of the system. The important feature of this approach is learning by experience from the environment which helps to start the process without prior knowledge of the system. In the area of cloud resource management, auto-scaler can act as an agent which interacts with the system components including VMs and physical machines. The state of the system is represented by the system attributes and performance indicators, while the reward is shown by the degree of QoS (such as response time or throughput) achieved by the application. The set of actions includes all possible corrective actions such as resource and application level reconfigurations to avoid the performance degradation. Dutreilh et al.[109] compare threshold based and Q-learning approaches for the problem of horizontal auto-scaling in the cloud. They investigate the functionality of each method in the presence of the main instability sources in the control systems, listing the observed potentials and weak points for each case. Duggan et al.[110] target the problem of VM live migration considering the available bandwidth and network congestion problem. They formulate the problem as an autonomous control system through utilizing RL and creating a multidimensional state/action space based on the VM utilization and available

bandwidth. VScaler proposed in [17] is another framework for the fine-grained resource management in the cloud which utilities RL to decide on the times to scale up/down in the system. To speed up the process of learning and exploration of the controller, they introduce the parallel technique which enables multiple agents to collaborate in different parts of the state space.

As we can see, the gradual learning concept provided by RL fits the nature of the problems in cloud performance management very well by involving the dynamism and uncertainty factors during the learning procedure. However, a main challenge that RL based solutions suffer is the size of the possible states and actions for the system. Considering the continuous nature of time series measurements and the scale of the target machines to be handled in large scale distributed environments, the problem of high dimensionality is becoming more important. To overcome this limitation, different approaches such as fuzzification of the table or using more abstract representation of data to limit the possible states or actions are proposed in the literature [68, 111]. Arabnejad et al.[111] extend a rule-based fuzzy controller with 2 different RL approaches, Q-learning and SARSA. The fuzzy concept helps to reduce the dimensionality of the state/action table which is an important issue affecting the complexity of the RL algorithms. Alternatively, a combination of dimensionality projections and sparsity based data structures are used in [112] to overcome the dimensionality problem. The proposed approach is used to manage efficient migration of VMs in real-time with respect to the energy and performance of the system. Adaptive partitioning of state space is another approach which initiates the model with one or few states and gradually decides on the partitioning of states during the interactions with the environment [113].

2.7 Action Trigger Timing: from Reactive to Proactive

When a corrective action should be triggered is a challenging question as it is highly dependent on the nature of the application and SLAs. Traditional approaches to this problem are mostly reactive. In *reactive methods*, any decision about the changes in the number of VMs, the configuration of resources or VM replacements is a response to the abnormal behaviour of the system indicators that identify changes in the performance

or QoS. As the degradation already has occurred and considering the delays before corrective actions take effect, an amount of the SLA violations should be allowed in SLA contracts. Moreover, in an unreliable environment involving various factors to affect the stability of the system, fluctuations in the performance are a common observation which can increase the number of occurrences of SLA violations. Usually, these approaches follow a threshold based strategy where the scaling starts after the measured metrics exceed an accepted value [4, 114]. For example, when the CPU utilization of server exceeds the threshold, new resources are added to the system. Therefore, reactive approach does not consider the performance anomaly as a gradually happening event with detectable pre-signs.

A step further in adaptive design of the resource management system is to include periodical actions to resolve possible performance problems at each time interval. While reactive approach performs resource adjustment action as a response to the performance degradations, periodical solutions are triggered by time. For example, software rejuvenation is a preventive technique that tries to clear the state of the software. This process can be repeated at regular intervals to resolve software performance degradations or avoid possible problems caused by software aging phenomenon [115, 116]. These time-triggered techniques can also be categorized as proactive approaches when regular updates of the system help to alleviate performance problems before they cause violation of SLAs. Alternatively, *event driven* proactive approaches perform corrective actions when an event is detected which is suspected to affect the behavior of the system in an undesired way. Event-driven proactive methods attempt to find the warning signs before they can cause unacceptable levels of performance degradations, so they can start preventive actions such as migrating VMs or adding new resources [18, 26]. They focus on the future events and are mainly based on the prediction of the future values and states. If proactive analysis of data can give an early enough alert of a possible performance problem, it helps RMM to plan and quickly start a proper action before the system goes into an anomaly state. Accordingly, the system returns or continues the normal condition, reducing the number of violations. One point worth mentioning here is how to decide on a proper value for the period of prediction. Short-term predictions are more accurate in the case of the workload related metrics because the measurements

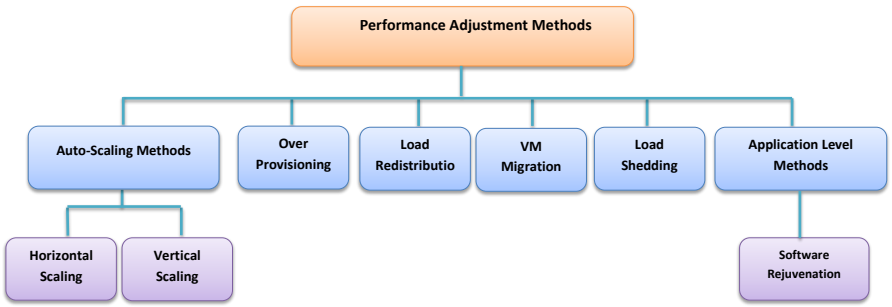


Figure 2.5: Cloud Performance Adjustment Techniques

close in time show higher correlations compared to the observations for longer in time. Longer time predictions are more challenging and better fit the data with regular patterns or seasonality. Di et al.[15] investigate long-term prediction problem for workload data in cloud data centers. They propose a Baysian-based method to predict the average load in the system, based on the derived features that capture different aspects of the statistical characteristics of data. Another strategy to decide on prediction interval is considering the required time for the corrective action to be effective in the system. If the workload is predicted for an interval shorter than the action time, the value of proactive resource management diminishes. Islam et al. follow this approach to determine a prediction interval based on the time it takes to launch a new VM. Therefore, upon receiving an alert of a possible load problem, the system has enough time to start the scaling action. Finally, *Hybrid solutions* are based on a combination of reactive and proactive approaches. These solutions are more realistic ones for the applications in the real environment where there is always the possibility of happening new and unknown events that are not detected by proactive mechanisms [18, 75]. In these cases, the reactive component helps the system to decrease the adversary effects of undetected anomalies in the system while the proactive module can learn the new undesired behavior by updating prediction models with recent observations.

Table 2.1: An overview of performance adjustment methods

Method	Contributing users	Advantage	Limitations
Application Level	CSP	No dependency on CRPs, No cost from newly added resources	This approach may not solve non-application originated problems and is not applicable for all types of the applications. It may cause short-term performance degeneration caused by non responsive components during deactivation time
Over-provisioning	CRP, CSP	Increases reliability for highly dynamic workloads	Resource wastages, Higher costs
Horizontal Scaling	CRP, CSP	No need for special hardware support	Increases all resource types without considering the source of the problem (homogeneous VM types), Start-up delays
Vertical Scaling	CRP, CSP	Custom resource adjustment (cost effectiveness), No need for new software instances (License problems)	Require special hardware supports
Load Distribution	CRP, CSP	No extra resources (cost effectiveness)	Limited applicability (especially for non-stateless systems)
Load Shedding	CSP	No extra resources (cost effectiveness)	Performance degradation by ignoring some requests
VM Migration	CRP, CSP	No extra resources (cost effectiveness), Reducing cold spots (energy saving)	Short-term performance degeneration during movement, Security/Privacy issues

2.8 Performance Adjustment Methods

Upon receiving an alert of an ongoing or possible performance problem in the system, RMM should start an adjustment process so the active resources meet the new requirements of workloads and applications. There are different solutions to alleviate the performance problems in the system including changing resource configurations, adding new computing units or replacement of the VMs. In this section, we explain these techniques more and especially focus on coarse-grained methods at resource and VM level as shown in Figure 2.5. Table. 2.1 presents an overview of some of the advantages as well as limitations of these methods. As explained in Section 2.4.1, CSPs are included as the contributing users for all types of decision making to highlight the importance of considering the application side SLAs regarding the related objectives such as budget and security constraints during the decision making process.

2.8.1 Application Level Methods

This category of actions is directly applied to the application and its environment, and therefore they are not specific to cloud systems. This approach targets the degradations in the performance of the application or operating system caused by the internal problems such as data corruption, numerical error accumulation or exhaustion of operating system resources [117]. Software rejuvenation is a possible corrective action in these types of the problems which tries to identify the problematic application or system component, clean its internal state and restarts the components. Indeed, simple application or VM restarts which are used by the system administrators as the first reaction to many performance degradations are preliminary cases of applying this approach. Dynamic component activation can also be triggered at this level, when the deactivation of optional components helps the system to manage higher loads or save energy according to the SLA requirements [53]. Since the application level methods directly affect the functionality of the application, CSPs or application owners are actively involved in making these decisions.

2.8.2 Over-provisioning

In order to efficiently utilize the capacity of cloud offered resources, it is vital to have a proper estimation of required resources that keep the performance and QoS of applications at an acceptable level. However, resource estimation is a complex problem especially for dynamic workloads with time-dependent fluctuations. A traditional solution to handle the uncertainty in resource demands of the applications is to provide enough resources to process the maximum expected workload in the system. This solution guarantees a stable application performance in the presence of workload fluctuations. However, the peaks in the workloads are usually transient events that do not last long and happen very rarely which means that most of the times resources are underutilized and incur extra costs for CSPs. Considering dynamic nature of the applications such as web workloads, over-provisioning is not avoidable in the fault-tolerant resource management solutions; However, researchers try to find a proper trade-off between the level of fault tolerance of the system and over-provisioned resources to have more control over the functionality of the system [42].

2.8.3 Auto-Scaling Methods

Scaling is defined as increasing or decreasing the amount of resources (ex: CPU, RAM, Disk, etc) for meeting SLA and performance standards. The scaling of resources can be done by changing the number of machines in the system or at the VM level by changing the configuration of one VM. In the following paragraphs, we explain each of these approaches in more details.

Horizontal Scaling

Horizontal scaling which is the primary block of every RMM framework helps to provide more resources by adding new VMs in the system [18, 114, 118]. The unit of changes in horizontal scaling is one VM. However, the newly added VMs can have a customized resource configuration (CPU, RAM, I/O) or pre-configured VMs can be launched by selecting one of the instance types offered by the provider. For example, Amazon offers

a range of VM types with different levels of configurations from small to very large instances [4]. As a complementary option, Google also offers users the chance to define their requirements with more details by launching customized instances.

The time it takes for a new VM to be launched, also known as VM start-up time, is highly important, especially for proactive RMMs. If start-up times are longer than the predicted point for a possible breach of the SLA, the effect of the scaling process will be same as the reactive strategies with the added overhead of data analysis module. Considering this, Mao and Humphrey[119] try to investigate possible influential factors such as the type of the instance, OS image size or VM location for different providers. The study provides researchers a basic understanding of the contributing factors which should be constant during the experiments and also estimated average times to be taken into account in the simulation of cloud environment [120].

Vertical Scaling

While horizontal scaling is a conventional strategy in the management of resources offered by providers, the advent of virtualization techniques has introduced new resource level scaling opportunities including elastic VMs. In vertical scaling, the VM elasticity feature is utilized to change existing VM configurations, virtual cores or RAM, on the fly to adapt to new requirements of the system [18, 121]. The online reconfiguration without turning the VM off is getting more attention especially when the time and cost factors are considered in RMM decisions. First of all, maintaining SLA objectives becomes challenging when there are sudden changes and spikes in the workload, and RMM needs a quick solution to increase the resources in the system. In horizontal scaling, the time it takes to launch new VMs can be a bottleneck when a fast effective solution is needed. On the other hand, new VMs require new software to be installed which can lead to additional license costs. Elastic VMs can offer the required concepts and technologies to implement practicable strategies for these problems. Indeed, depending on the application, resource level scalings might be the most fitting idea in the case of resource shortages. For example, a CPU saturated system hosting a CPU intensive application can be scaled by adding a new core to the VM without wasting memory, bandwidth and

other resources. Moreover, the same VM can continue the execution of existing requests without a need for interruption or transferring their execution profiles to a separate VM.

However, the elastic VMs need the support from both the guest VM OS and hypervisor. Therefore, due to added complexity, many public providers do not offer this functionality. [9] is one of the Infrastructure as a Service (IaaS) providers that supports the live vertical scaling of CPU cores, RAM, Network Interface Card and Hard disks. To have a better understanding of the extent of support for vertical scaling, one can refer to work done by [122] which studies this capability for some of the common hypervisors and guest OS as well as OpenStack framework in the cloud environment. It should be highlighted that the capabilities of vertical scaling for the VM is limited to the physical host. This means that the amount of resources which can be added to the VM can not be more than the available resource on the physical host. Therefore, in the case of resource shortage on the host, a new host with the available amount of resources can be selected and VM migrations precede vertical scaling to provide required resources for upgrading VM configuration.

2.8.4 Load Distribution

In a large scalable environment, we can find many replicas of one service such as database or application server components. The problem of distributing application requests among existing replicas of one component is a functionality provided by load balancers. A load balancer such as haProxy can be configured with a weight for each replica and distribute the loads according to this configuration among multiple VMs. Amazon Web Service Elastic Load Balancer (AWS ELB) [4] offers a simple round-robin strategy which assigns an equal weight for all active VMs and selects them from a list in the order of appearance. Therefore, in a round-robin based balanced environment, the utilization of all VMs is affected similarly. Noticeably, this approach does not provide a cost or energy efficient solution especially in an underutilized environment, where a small fraction of the provided computing and storage resources are enough to maintain the required SLA. A more efficient version of this approach assigns weights based on the specific server characteristics such as the load of the server which helps to send new requests

to least utilized servers first [123]. Grozev and Buyya[120] follow this approach to consolidate web requests in a few servers without violating the QoS. The proposed method monitors CPU and RAM utilizations of the servers along the availability of the network buffer capacity to assign new requests. Soni and Kalra[124] prioritize the servers based on their hardware characteristic, distributing loads based on the computation power and availability of the machines. These approaches help the system to balance the workload among existing resources to keep the performance at the acceptable level. However, load distribution is a task level resource management which does not control the amount of resources in the system. Therefore, they should be integrated with other scaling methods which help to maintain enough resources to handle the existing workload of the application.

2.8.5 Load Shedding

Load shedding is a type of self-healing approaches, primarily used in electrical power management to handle high loads in the system. The idea behind this approach is to maintain the availability of the system by sacrificing some quality of service in the presence of faults. In data stream mining, load shedding refers to mechanisms that try to find when and how much of data can be discarded so the system can continue working with acceptable degradation of performance [125]. This approach is not effective for the applications with highly dynamic workload and strict QoS requirements [126]. However, one can consider request admission and resource reservation policies in the system in terms of the SLA agreements so the extra requests that can not be handled by available resources will be rejected, saving time and money for both users and service providers.

2.8.6 VM Migration

The scalability feature of cloud systems is highly dependent on virtualization technology that enables multiple applications to reside on one physical machine by sharing available resources. VMs are preferably not dependent on one machine and can be moved among different hosts when needed. This capability brings new opportunities for im-

proving the utilization of cloud applications by finding proper placement of VMs in the system [127]. Migration can be done to fulfill different objectives. Migrating VMs from underutilized hosts and consolidating them in few hosts enable the system to shut down unused hosts, saving costs and energy. Duggan et al.[110] investigate this problem while considering the available bandwidth as a factor to determine the best time for migrations. An over-utilized host, where the guest VMs are consuming all the available resource offered by the host, is another target which can benefit from migration. In this case, the application on the overloaded host may experience performance degeneration as there are requests that can not be handled in time due to the saturated resource and long waiting queues. Accordingly, Sommer et al.[128] propose a prediction based migration strategy to find the overloaded hosts and triggers migration procedure to move some of the VMs to the existing underloaded hosts. A combination of multiple forecasting methods is employed to predict the future resource consumptions of the VMs and identify the possible overloaded hosts based on the predicted values.

Migration is also a complementary procedure when the host can not fulfill the requests to increase the resource capacity required for vertical scaling actions. PREPARE proposed in [26] utilizes this strategy to correct the performance problems caused by internal faults or load anomalies. The live migration is called when the vertical scaling action is ineffective or not possible due to the lack of the resources on the host.

2.9 Gap Analysis and Future directions

Based on the analysis of different aspects of performance-aware resource management in clouds, a number of challenges have been observed that needs to be investigated more thoroughly. Accordingly, we propose a list of the potential research areas and future directions in the following sections.

2.9.1 VM Elasticity Analysis

Vertical elasticity is one of the rather new functionalities introduced for cloud resource scaling which has not yet been prevalent compared to horizontal scaling. Technical limi-

tations to support VM elasticity along with the higher complexity of fine-grained scaling (CPU and RAM level compared to VM level) management in data centers makes this approach limited in practice and most known public providers do not offer this service. Therefore, a more detailed study of the characteristics of vertical elasticity especially time sensitivity analysis for different resource types, OS and hypervisors is required to help the design of accurate solutions in the area of proactive resource management.

2.9.2 Adaptable Learning in Cloud

The effectiveness of many advanced machine learning methods is highly dependent on pre-configurations that define a proper threshold or parameter values, usually based on the characteristics of data and applications. There are many works targeting the problem of parameter configuration of learning methods in the field of data prediction and anomaly detection. However, prediction and control of system behavior is becoming more complex with the growth of highly dynamic environments such as cloud. Considering these challenges, robust statistics metrics and model/assumption free frameworks are among techniques which require more attention to have more realistic solutions. These mechanisms help to increase the flexibility and robustness of models which is highly important for improving the adaptation of systems to a variety of application and datasets. Dynamic tuning of model parameters in accordance with recent changes and feedbacks of the system are interesting approaches for developing robust solutions to be applicable for different workload patterns. Self-adaptable systems which are discussed in section 2.6.2 and gradual learning frameworks such as reinforcement learning from Section 2.6.2 are among approaches that can be effectively combined with the data learning techniques to enable a mechanism of live interactions with environment to have adaptable resource management decision makings. However, a common point of failure for these mechanisms is the exploration capability in terms of dealing with large state/action tables which affect the accuracy and convergence rate of the solutions. While new approaches such as adaptive partitioning of state space [113] are introduced that address these types of the problem, the applicability and efficiency of proposed solutions should be more investigated.

2.9.3 Anomaly Cause Inference

Anomaly cause inference has been studied as part of the data analysis module to help RMM better decide on selecting corrective actions. However, existing works in this area mostly are coarse-grained, giving suggestions about the bottleneck metrics based on the resource level information. Moreover, the contribution of knowledge from cause identification in the process of RMM planning and resource management has not been fully investigated. A more autonomic and integrated cause identification process which has continuous interaction with planning module to get timely feedbacks on the quality of information is a challenge for future researchers.

2.9.4 Application Dependent Detection Accuracy Trade-offs

In data analysis part of the investigated problem, Area Under the Curve (AUC) commonly is used to show the performance of anomaly detection algorithms in detecting anomaly points. However, in the area of cloud performance analysis, the number of normal instances in the collected data usually is much higher than anomaly points. The lack of the balance in the number of instances for different classes raises the question of whether AUC metric is biased by true negative points. We believe that presenting the performance results by comparing both metrics AUC and Precision-Recall Area Under the Curve (PRAUC) which demonstrate the functionality of the algorithms from different points of the view is an important part of the anomaly detection problems in this area. This is a point that can be very important for some applications which require complex recovery points in the case of the true anomaly events. For example, for prevention mechanisms that target disk related problems with expensive mitigation actions, a solution with higher precision and minimum of false alarms may be preferred. Referring to our survey, this is an interesting point which is highly neglected and requires a detailed analysis of the effectiveness of proposed anomaly detection methods considering service owner preferences.

Table 2.2: Comparison of Data Aware Performance Management Approaches in Large Scale Systems

Work	Data Level	Learning Approach (ML: Machine Learning, RL: Reinforcement Learning)	Anomaly Aware	Anomaly problem	Cause Inference Level	Proactive	Resource Adjustment Techniques (H: Horizontal, V:Vertical)	Module
[15]	System	ML	X	_	_	✓	Load balancing	Data
[16]	System	ML, Statistical	X	_	_	X	_	Data
[18]	System	Threshold, ML	X	_	_	✓	V, H	Data, Plan
[114]	System, Application	Threshold, Statistical	X	_	_	✓	H	Data, Plan
[21]	System, Application	ML	✓		_	X	_	Data
[22]	Network	Statistical	✓	Network	_	X	_	Data
[30]	System, Structure	ML	✓	Software bug, Resource bottleneck	Component, Metrics	X	_	Data
[20]	System	ML	✓	Resource bottleneck	Type of Anomaly	X	_	Data
[25]	System, Application	ML	✓	Resource bottleneck	Type of Anomaly	✓	_	Data

[23]	System	ML	✓	Deadlock, Starvation, Livelock	Type of Anomaly	✓	_	Data
[27]	System	ML	✓	CPU/Mem leak, Network hog	Metrics	✓	_	Data
[26]	System	ML	✓	Resource bottleneck	Metrics	✓	V, Migration	Data, Plan
[68]	System	RL, ML	X	_	_	✓	H	Data, Plan
[40]	Network	Signature, ML	✓	Network	Type of anomaly	✓	_	Data
[28]	System, Application	ML, Statistical	✓	Resource bottleneck	Metrics	X	V, Migration	Data, Plan
[29]	System, Application	ML, Statistical	✓	Resource bottleneck	Metrics	X	_	Data
[31]	System	Statistical	✓	Resource bottleneck, Offload bug, Load balancing bug	Component	X	_	Data
[32]	System	Statistical	✓	Resource bottleneck, software bugs, multi-tenancy problem, network packet loss, deadlock	Type of Anomaly (external vs internal)	X	_	Data
[33]	System	ML, Signature	✓	Software bugs	Code level	X	_	Data

[34]	Application	Statistical	✓	Software bugs	Code level	X	_	Data
[35]	System	ML		Resource bottleneck, Database abuse	Target Thread/Process	X	_	Data
[65]	System	ML, Statistical	X	_	_	✓	_	Data
[41]	Network	Statistical	✓	DDos Attacks	_	X	_	Data
[63]	System, Application	Statistical	✓	Resource bottleneck	Metric	X	_	Data
[71]	System, Application	Signature	✓	_	_	X	_	Data
[72]	System, Application	Statistical, ML, Signature	✓	Resource bottleneck, Application update	_	X	_	Data
[73]	System, Applica-tion, Structure	Signature, Statistical	X	_	_	X	_	Data
[74]	System, Application	Statistical, Signature, ML	✓	Load, Software bugs	Metrics, Type of Anomaly	X	V, H, Migration	Data, Plan
[75]	System	Threshold	_	_	_	✓	H, Migrations	Data, Plan
[76]	System, Application	Threshold	X	_	_	X	H, V	Data, Plan

[91]	System, Application	Control Theory	X	_	_	P	H	Data, Plan
[77]	System	ML	✓	Resource bottleneck	_	X	_	Data
[86]	System	Control Theory	✓	Load, Hardware failure	_	X	H	Data, Plan
[81]	System, Application	Control Theory	X	_	_	X	V	Data+Plan
[82]	System, Application	control Theory	X	_	_	X	H	Data+Plan
[96]	System	Statistical	✓	Resource bottleneck, Software bugs	_	X	_	Data
[97]	System	Statistical	✓	Resource bottleneck	_	X	_	Data
[98]	Application, System	Threshold, Statistical	✓	_	Metrics	X	_	Data
[99]	System, Application	Threshold, Statistical	✓	Resource bottleneck	Metrics	X	_	Data
[100]	System, Application	ML, Statistical	X	_	_	✓	H	Data, Plan
[101]	Network	Statistical	✓	Port scan	Packet Information	X	_	Data
[102]	System	Statistical	✓	DoS Attack	_	X	_	Data

[104]	System, Application	ML	X	_	_	✓	_	Data
[105]	System	ML	✓	Resource bottleneck	_	✓	VM Placement	Data, Plan
[109]	System, Application	Threshold, RL	X	_	_	X	H	Data, Plan
[17]	System	RL, Statistical	X	_	_	✓	V	Data, Plan
[110]	System	RL	X	_	_	X	Migration	Data, Plan
[111]	System, Application	RL	X	_	_	X	H	Data, Plan
[42]	System	Signature	✓	Resource Shortage	_	X	H, Over-provisioning	Data, Plan
[129]	System, Application	Signature, ML	✓	Resource bottleneck	_	✓	V, Migration	Data, Plan
[121]	System	Threshold	✓	Resource bottleneck	_	X	V, Migration	Data, Plan
[127]	System, Application	Threshold, Statistical	✓	Resource bottleneck	_	✓	Migration	Data, Plan
[128]	System	Statistical	X	_	_	✓	Migration	Data, Plan
[38]	System	Rule	✓	Network related Application bugs	Target System Calls	X	_	Data
[4]	System	Threshold	X	_	_	X	H	Data, Plan

2.10 Summary

This chapter investigates different approaches in the performance management of cloud environment. Identifying the major limitations and considerations in these approaches and their impacts on the selection of the best strategies for proper resource configuration highlights the need for more advanced and automate procedures to handle the dynamism of the environment. We have proposed a taxonomy of problem focusing on the value of the data as a source of knowledge for resource management decision making and presented a survey of the existing works, accordingly. The listed categories in the taxonomy are defined based on the characteristics of the existing works within the scope of this thesis and include their base architecture, granularity of collected performance data, targeted performance problems and the types of resource management actions. Based on the reviewed works, a list of observed gaps and possible directions is discussed which can give new insights for further research in this area. In the following chapters, we present our research contributions in this area addressing some of the discussed challenges and gaps in this chapter.

Chapter 3

Performance Anomaly Detection Using Isolation-Trees in Computing Clouds

In order to efficiently manage resources in cloud, continuous analysis of the operational state of the system is required to be able to detect performance degradations and malfunctioned resources as soon as possible. Every change in the workload, hardware condition or software code, can move the state of the system from normal to abnormal which causes performance and quality of service degradations. These changes or anomalies vary from a simple gradual increase in the load to flash crowds, hardware faults, software bugs, etc. This chapter addresses the first research question introduced in Section 1.2 by proposing an Isolation-Forest based anomaly detection (IFAD) framework based on the unsupervised Isolation technique for anomaly detection in a multi-attribute space of performance indicators for web-based applications. We empirically validate the effectiveness of proposed technique with regard to various workloads and anomaly types which shows that IFAD can achieve good detection accuracy especially in terms of precision for multiple types of anomaly.

3.1 Introduction

The emergence of cloud service providers (CSPs) such as Amazon, Google and Microsoft has moved the previously limited, community specific capabilities of high performance computing to a new era of public, on-demand, pay-as-you-go computing. These new

This chapter is derived from:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, Performance Anomaly Detection Using Isolation-Trees in Heterogeneous Workloads of Web Applications in Computing Clouds, *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 31, No. 20, ISSN: 1532-0626, Wiley Press, New York, USA, Oct 2019.

characteristics offered by cloud providers mainly highlight the need for more complex and robust resource management solutions that decrease the need for human involvement. The main goal for CSPs is to find better ways of resource management to improve resource utilization and guarantee the quality of service (QoS) experienced by their customers. Any violation of these service level agreements (SLA) can cost providers penalties for SLA violations or losing their reputation. However, considering the dynamic nature of cloud systems, every change in the workload, hardware condition or software code, can change the state of the whole system from normal to a state of abnormal behavior which can affect the performance and QoS. The degradations in the performance can result in higher monetary costs and energy wastage for under-utilized resources which are negative sides of the dynamic environment from resource provider perspective, highlighting the need for automated ways of detecting performance problems [7]. This is a highly important observation, especially for large scale web application systems where the interaction from users to web servers can change frequently, affecting the pattern of workloads and resource requirements. For example, it is shown that web applications are prone to many of the performance problems which involve CPU and memory resources [8].

Taking into account that each type of performance problem can impact the system or application metrics differently, defining proper rules that cover all types of problems is becoming complex and out of the expected knowledge of application owners. It is vital for every resource management solution to consider utilizing timely and adaptive algorithms to identify the anomalies in the system as soon as possible. Therefore, researchers are looking for more powerful solutions for performance analysis of resources in the cloud. An automatic anomaly detection module should be able to analyze the collected performance metrics from cloud resources and build models which can detect deviation points where the system moves to an anomaly state. In the process of collecting metrics, building models and triggering alerts, there are some challenges that should be considered:

- **Scalability**: One of the main characteristics of cloud dependent applications is scalability which makes it possible to scale up the system components to hundreds and thousands of virtual machines (VMs). In such a dynamic environment, cen-

tralized anomaly detection approach becomes a problem especially if we want to capture the state of the whole system in one model. As a solution for this problem, we assume that each machine monitors the performance metrics of its own VMs which breaks down the problem of anomaly detection to one host. Furthermore, we are utilizing an easy to deploy monitoring tool, known as Ganglia that can be easily managed in a large distributed environment.

- **Unsupervised learning**: Cloud environment is prone to different types of anomalies that affect the performance of the system in different ways. Therefore, it is reasonable to assume that we do not have access to labels that identify the state of the system as normal or anomaly. Accordingly, the proposed anomaly detection module does not assume prior knowledge of the system and would perform in an *unsupervised* manner.

- **Recurrent model parameters tunning**: Due to the heterogeneous nature of web applications in the cloud, the normal state of the system can change significantly based on the number of requests sent to the system. In this case, detecting anomalies in a previously unseen normal environment is another challenge that we should consider. Most of the existing algorithms require tunning and parameter settings to be done before updating the models. This procedure adds extra overhead to the system, particularly for frequently changing environments. The proposed anomaly detection approach is fast which requires no workload dependent configuration or any time consuming data preparation which makes it a fit for our target environment.

- **Application preferences**: The problem of anomaly detection is highly application and data dependent. We need to consider cases when the application owner prefers an algorithm with a higher precision, sacrificing the sensitivity of the algorithm or vice versa. For example, applications concerning disk drive failure analysis or medical tests for rare diseases require low and more controlled false alarms rates considering the costs of triggered actions for predicted anomalies [130]. Hence, in this work, we evaluate the effectiveness of the proposed IFAD framework with different algorithms in terms of both measures of AUC (Area Un-

der the Curve) and PRAUC (Precision-Recall AUC) and the trade-off between false negative and positive rates in the results. The study helps to better understand the capabilities of algorithms from the perspectives that are usually ignored in current research.

With regard to these challenges, this chapter focus on the first two phases of MAPE loop (Monitoring and Analyzing) as discussed in Section 2.2.1 and investigates an unsupervised anomaly detection approach for analyzing different types of anomalies (CPU, memory, disk, ...) in heterogeneous workloads of web applications in the cloud. We deploy a realistic prototype for Web2.0 applications based on the CloudStone benchmark and integrate that with an injection module by implementing five types of performance anomalies in cloud environments. The contributions of this work are therefore, a time-series based anomaly detection module that can handle various types of web workloads in terms of trend and seasonality features in the presence of performance anomaly problems. Especially, through our experiments, we show that analyzing the performance results by comparing both metrics AUC and PRAUC, which demonstrate the functionality of the algorithms from different points of view, is an important part of the anomaly detection problem that should be further investigated. Moreover, interesting characteristics of Isolation-Trees based anomaly detection to offer a low overhead algorithm with a simple yet effective procedure makes it a new alternative to analyze other types of anomalies or applications.

The rest of this chapter is organized as follows: Section 3.2 reviews some of the existing works in the field of performance management and anomaly detection. Section 3.3 presents the motivation and an overview of the main parts of IFAD framework. In Section 3.4, we detail the functionality of each part including characteristics of the collected data followed by data processing and finally anomaly detection module. The details of all experiments and the results are presented in Section 3.5 and finally, we summarize the work and findings in Section 3.6.

3.2 Related work

In this work we have proposed an Isolation based anomaly detection framework to detect performance anomalies in the 3-tier web-based applications and investigated the effectiveness of multiple algorithms based on AUC, PRAUC and DET measurements. This is a starting point to give insights to system administrators about the importance of the specific requirements of their application in selecting a suitable data analysis approach. In this section, we first discuss anomaly detection algorithms in general and then focus on the anomaly detection applications in cloud environment.

3.2.1 Anomaly Detection

The concept of anomaly detection has been widely studied under different names outlier or novelty detection, finding surprising patterns or fault and bottleneck detection in operational systems. There are a variety of survey and review papers that try to classify existing algorithms based on their requirements and computation approach into different categories[19, 131]. Distance based algorithms utilize an approach that addresses the problem of outlier detection based on the concept of the distance of each instance to the neighborhood objects. Greater the distance of an instance to the surrounding objects, more likely that the instance is an outlier [132]. Another approach defines the local density of target instance as a measure for the degree of outlierness of that instance. Objects that reside in the low degree regions are more likely to be known as an anomaly [133, 134]. While distance and density based approaches show promising results in various types of the datasets, they usually require complex computations which are not preferable in the high dimensional or fast-changing environments.

Another anomaly detection approach which demonstrates promising characteristics in terms of the time complexity and memory requirements is isolation-based technique [107]. In contrast to the traditional approaches of anomaly detection that anomalies are detected as a by-product of another problem such as classification and clustering, isolation-based technique directly targets the concept of the anomalies based on the idea that an anomaly instance can be isolated quickly in the attribute space of the problem compared to the normal instances. This approach has been also explored in other types

of the applications such as fraud detection problems. For example, [135] addresses the categorical values and proposes an isolation based anomaly detection based on the horizontal partitioning of the data. They show that the proposed method can detect some of the hidden anomalies in the subsets of data that can be ignored when the whole data is analyzed. However, their method is highly domain specific and needs pre-knowledge of the structure of datasets. Another work by [108] proposes a sequential feature selection and outlier scoring framework which tries to filter the important subset of features. An outlier scoring algorithm calculates the scores and they try to find a regression formula among outlier scores and original features as the predictors. They have also demonstrated their approach based on the isolation technique as an outlier scoring algorithm and have shown the effectiveness of proposed filtering approach in the high dimensional data. In contrast, our approach utilizes time series analytics and isolation based technique for detecting bottleneck anomalies in the cloud hosted web applications.

3.2.2 Anomaly detection in cloud

The idea of using anomaly detection to find faults in the computing and storage systems has been widely investigated. For example, [130] studies specific requirements of disk performance analysis to have a controlled false alarm, proposing improvements on existing algorithms to avoid high penalties during the disk failure analysis. Hence, they propose statistical testing based approaches and multivariate decision rules to predict disk failures with the aim of reducing false alarms in the prediction process. [63] studies the application of tree-augmented Bayesian networks (TAN) classifiers to relate the resource performance metrics to SLO violations for web-based applications. Although they investigate the effect of different workloads and SLO thresholds, their work does not compare TAN performance with other learning algorithms and neither studies the PRAUC or DET metrics as our work. The work presented in [136] investigates the feasibility of isolation technique to detect anomalies in the data from IaaS datacenters. However, their focus is on the behavior of IForest in the presence of seasonality/trends in their dataset and they do not consider types of anomalies or compare the detection capabilities of IForest with other algorithms for different performance problems and with

a variety of workloads.

[30] addresses the fault localization problem in distributed applications. The proposed framework combines the knowledge of inter-component dependencies and change point selection methods, taking into account that the abnormal changes usually start from the source and propagates to other non-faulty parts based on the interactions of the components. Principal component analysis is another method to analyze the data, especially to reduce the dimensionality of attribute space. Accordingly, [20] presents an automatic anomaly identification technique for adaptively detecting performance anomalies by proposing an idea that a subset of principal components of attributes can be highly correlated to specific failures in the system. In contrast, our work focuses on unsupervised bottleneck anomaly identification and can be used complementary to these works to detect previously unseen anomalies.

[99] addresses the problem of bottleneck and cause diagnosis by finding the correlation among attributes and application performance metrics. A subset of correlated metrics is selected based on the predefined thresholds and is analyzed to find possible causes of performance anomalies which are injected in the simulated data. However, the proposed approach is sensitive to the degree of temporal correlation among attributes. [137] targets the security issues that can arise after migrating VMs to new hosts. They propose a combination of an extended version of Local Outlier Factor (LOF) and Symbolic Aggregate ApproXimation (SAX) to detect and find possible causes of anomalies. The SAX representation helps LOF to consider the time information during analysis. However, LOF is a semi-supervised algorithm which is sensitive to the presence of anomaly in the training data. [114] applies a threshold based approach for the problem of resource management in web applications. The proposed framework starts to add new resources as a response to detected anomalies based on the observed violation of Response-Time or CPU utilization; moreover, a regression-based predictive algorithm method detects over-provisioned resources to be released. The work presented by [16] considers a single attribute, number of required processors at a certain time, for the resource utilization estimation. They propose a combination of machine learning and statistical methods based on the idea that the former is more reliable in long-term prediction whereas the latter can have more accurate predictions for the short-term in-

tervals. However, their prediction does not include the concept of unexpected behaviors resulting from various anomaly sources. Compared to these works, our work is more general in terms of considering richer feature space and other sources of unexpected behavior.

The application of unsupervised Hidden Markov Models to detect cloud performance anomalies is investigated by [77]. They propose a distributed and online anomaly detection framework, focusing on the 3 main attributes of Memory, CPU and disk. Our work, in contrast, targets higher dimensional problems with large number of features and therefore needs faster detection solutions with less computation complexity and adaptation requirements. [35] exploits unsupervised clustering to detect anomaly patterns at the thread and process level. They collect system level metrics based on the application characteristics and utilize DBSCAN method to detect non-normal behaviors. However, their method requires an off-line clustering of the normal data before starting the anomaly detection process.

[25] investigates proactive anomaly detection in data stream processing systems. The target anomalies are injected and the training phase is done on a labeled dataset of different anomaly occurrences in historical data. [26] addresses the same problem by integrating a 2-dependent Markov model as a predictor and TAN for anomaly detection. They utilize TAN models to distinguish normal state from the abnormal ones as well as reporting the most related metrics to each type of the anomaly. These works follow a supervised approach, targeting stream processing applications.

In contrast to existing works, our approach investigates the unsupervised anomaly identification in a multi-attribute space for heterogeneous workloads of web applications. Moreover, we highlight the importance of workload characteristics as well as application specific requirements by studying the relation of different measurements obtained from anomaly detection process.

3.3 Motivation and System Overview

The virtualized environment of cloud models is prone to various types of performance problems that make the resource management solutions more complex and challeng-

ing. Any change in the configurations of the resources is a response to a performance degradation or SLA violation which is influencing one or a group of related VMs hosting the application. This is an important issue, especially for web applications in which the behavior of users plays an important part in the performance of the application. For example, the sudden surges in the number of users due to the release of a new product requires an immediate response by adding enough resources to handle the requests. Resource management modules should be aware of these violations to take corrective actions immediately.

On the other hand, many existing anomaly detection techniques are not optimized for the highly changing conditions of the heterogeneous applications hosted on the cloud. A large number of clustering and classification techniques are based on the notions of distance and neighborhood density of instances in the feature space of the problem which leads to high computational complexity and memory requirements. The problem becomes more important in the area of cloud performance analysis, where a range of performance indicators from application and system levels can be affected by various types of faults in the system. The more features to be included during analysis, the more complex feature space with higher dimensionality. Moreover, the primary function of these algorithms usually targets other problems such as clustering and classification and anomalies are detected as a by-product of data categorization while Isolation technique directly addresses the anomaly detection problem [19, 107].

Regarding the problem of web performance analysis in the cloud, heterogeneity of workloads and different types of possible failures are two contributing factors that can make the above mentioned challenges even more important. Therefore, the first problem is identified as providing a framework that can quickly and effectively model different patterns of workload variations which include a signature of some of the common bottleneck problems. The second addressed problem in this work is the lack of comprehensive study on the effectiveness of various algorithms with regard to the precision of detection results. We have observed that the most common evaluation metrics discussed in the literature focus on the detection accuracy in terms of the true positive points. However, the results of corresponding metrics can be misinterpreted in the problems with highly imbalanced classes which highlights the need for other types of evaluation.

Figure 3.1: 3-tier Web Layers

Motivated by the above limitations, we formulate the problem as detecting unusual behaviors that may be a sign of the beginning of an abnormal state in the system with the possibility of changes in the normal patterns of the workload. We show how data analytics techniques can help to overcome the complexities of time series to better capture the pattern of data and anomalies. Regarding this definition, it should be clear what types of abnormal behavior are targeted. Given one VM measurement, we focus on a category of performance problems known as the resource bottlenecks. We try to find point anomalies which are measured records that are not consistent with the previously seen measurements of performance indicators of the system. As an example, consider a situation in which a memory-intensive background application starts working in the same VM with the target application. System observes an increase in the memory related metrics while the incoming workload of the application does not show any significant change. The unusual increase of the memory should be detected as an abnormal behavior as it can cause a performance degradation in the application due to the lack of enough available memory for handling user requests. Similarly, we consider the impact of the unusual increases in CPU, disk and load of the system. It should be noted that the primary focus of this work is to detect unexpected behaviors in terms of the variations of system performance. Therefore, the abnormalities in the resource utilizations are considered as the signs of application performance degradations which is consistent with the previous observations [8]. Moreover, our work concerns a data centric analysis of performance anomalies which do not rely on user perceived measurements such as response time and considers the violations in input data (collected metrics) as possible signs of misbehaving in the system which causes user side SLA violations. This is a common approach in studies that investigate the performance of anomaly detection techniques in cloud systems [77].

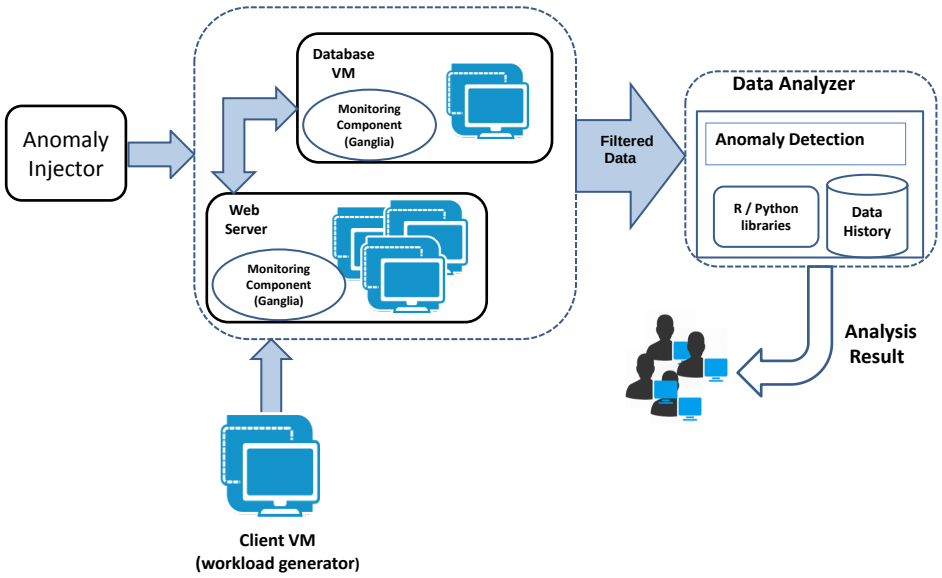


Figure 3.2: A High Level System Model

Accordingly, two main parts of the IFAD framework are identified as data preparation modules for each VM to analyze the specific characteristic of the time dependent datasets including trends/seasonality, and the anomaly detection modules. The target application in our model is a 3-tier web-based application which is a standard architecture employed by many web applications [138]. As shown in Figure 3.1, in this architecture, the functionality of the application is divided into 3 layers:

- Presentation Layer: The interface that user interacts to access the application layer.
- Application Layer: The main parts of the application/business logic are implemented in this layer.
- Data Layer: The persisted data that can be accessed by application layer is managed by one or more database servers in this layer.

It is shown that this type of applications is prone to many of the above mentioned performance problems [8]. The main focus of this work is the performance of the server

side VMs which can be easily affected by the behavior of users, buggy codes or other malfunctioning applications.

Figure 3.2 depicts a high level model of the proposed framework. In order to be able to investigate the effect of different performance changes in the system, a prototype in a virtualized environment is deployed. The prototype models the components of 3-tier applications which are common, yet simple to be manipulated for anomaly injection parts. A set of workloads with different configurations is generated to simulate normal state of the system. Consequently, different types of anomalies are injected into the benchmark components to show how each type of the anomaly can change the state of the system from normal to abnormal.

The VM monitoring module continuously tracks the performance indicators on the VM that is hosting the application. Every VM monitors its own performance metrics at regular intervals and sends them back to the anomaly detection module. The per-VM monitoring policy enables the system to quickly isolate the source of the problem to the target VM and focus on the identification of underlying reason in the machine. This type of problem may be ignored by system-wide solutions that monitor the average values of performance indicators in the system.

The collected metrics are either system related such as CPU, free disk and memory or application specific metrics such as SQL database attributes. Table 3.1 presents a list of some of the major attributes collected in our system. One point worth mentioning here is how to select a proper logging interval for monitoring modules. Selection of a proper log time is highly dependent on the nature of the workload, type of the application and reliability of the environment. Existing works select a range of values from a few seconds to hour(s) based on the target application and problem [16, 26, 77]. In our case, Ganglia gives us the flexibility to select any interval and based on our experiments we have used a configuration of 15 seconds interval to collect the values for all monitored attributes to have a desirable trade-off between storage and accuracy in the system. Collected measurements are processed by different filtering techniques before they are sent to the anomaly analysis module.

The aim of monitoring performance indicators is to detect any abnormality compared to the past observations. Upon receiving the filtered measurements for each VM,

Table 3.1: Some of the monitored metrics in the Application or Database servers. In total, there are 98 metrics collected from the monitored machines.

Metrics	Description
CPU	CPU utilization
CPU_IO	The time waiting for IO operations
MEM_FREE	Available memory
MEM_DIRT	Memory waiting to be written to disk
DISK_READ	The time in seconds spent reading
DISK_WRITE	The time in seconds spent writing
BYTES_IN	Number of bytes received
BYTES_OUT	Number of bytes sent
ABORTED_CONNECTS	Failed connection attempts to MySQL
THREAD_CONNECTED	Currently open connections to MySQL

the system needs to have an abstract model of the past behavior as a reference for the comparison. Anomaly detection module has the responsibility to initialize and update the models by training target anomaly detection algorithm with the past observations. Therefore, the core part of this module is an algorithm which generates the models by learning from training data and applies them for detecting anomalies in the test data. IFAD leverages isolation based approach for this problem which is detailed in Section 3.4.4. The output of this module is the anomaly scores for new observations. One can use the obtained knowledge from anomaly detection module to improve decision making procedure of resource management frameworks to start proper mitigation actions or alleviate performance degradations in the system. This can be done by various preventive actions such as changing resource configurations or scaling resources which are discussed in the next chapters.

3.4 System Design

In this section, we explain the data collection and preparation procedure followed by the details of the anomaly detection module.

3.4.1 Data Collection

In order to have a better understanding of current state of the system and possible drifts to abnormal states, two sources of data are considered: 1) Application data which is workload and component related 2) System data which represents the current state of the resources by combining different metrics such as CPU utilization, memory and disk usage. However, there is no public dataset that provides all the required measurements from application characteristics to VMs performance states. To be able to validate the anomaly detection procedure we need to have a labeled dataset which shows the exact points where an anomaly occurs. Moreover, the source of anomalies can be different. Since we want to have a signature of each type of the anomaly, different anomalies such as CPU or memory bottlenecks should be injected randomly in time in the system. To address these issues, the analysis is performed by generating typical workloads through a standard web benchmark based on CloudStone framework [139]. Each component is installed on a separate VM and the proposed framework is deployed on Australian virtualized cloud infrastructure named Nectar.

To collect the performance indicators of the components, Ganglia framework which is a scalable, distributed monitoring system is utilized . Ganglia is based on a hierarchical architecture with a multicast-based send/receive protocol to monitor the state of the machines in a cluster and aggregates the monitored states in a few representative nodes. The robust design of the data structures and algorithms helps us to easily extend the functionality of the framework by adding new scripts to collect customized measurements from monitored components. For example, the RAM related metrics provided by Ganglia does not include the *active memory* metric. *Active memory* shows the amount of memory recently used considering the fact that some parts of the allocated memory

https://nectar.org.au/research-cloud/
http://ganglia.info/

Table 3.2: The range of CPU utilization for each workload level

Workload Level	CPU Utilization
Low	10% - 40%
Medium	40% - 60%
High	60% - 100%

which are included in other metrics such as *consumed memory* are not being used by the application and are set to be freed. Therefore, *active memory* is a more accurate estimation of RAM utilization of the application in each time interval. We have extended the basic monitoring module of Ganglia by adding the scripts to calculate this value in our installation. Table 3.1 shows a list of some of the major attributes collected in our system. The framework collects data in RRD (Round Robin Dataset) format and sends the collected files to the analyzer module. Data analyzer module can read monitored performance data from these files and perform data preprocessing steps before applying the detection algorithms.

As we already stated, the increase/decrease in the number of users interacting with web application can highly impact the pattern of the workload and the utilization of the resources. In order to have a comprehensive validation of the effect of possible trends in the experiments, five types of the dataset are generated by changing the frequency of increase in the number of concurrent users in the system. The changes in the number of users can happen at the start of each step which corresponds to one run of the benchmark. For reproducibility of data by other researchers, we annotate each dataset with the level of resource consumption from the starting point to the end. Having these annotations, we can regenerate the datasets on machines with different specifications. We define three levels of the number of users based on the observed resource utilization during different experiments on the benchmark. Since the target workloads are more CPU intensive, we correspond each level to a range of CPU utilization for the application server. Table. 3.2 shows three ranges of CPU utilization for these levels. The details for each dataset are as follows:

Dataset1: The number of concurrent users in the system is medium. Therefore none

of the resources is overloaded and there is plenty of free CPU and Memory space. The frequency of changes in the number of users is very low, so it simulates a workload without any load related fluctuations, and anomalies show a distinguishable pattern in all parts of the data.

Dataset2: The number of concurrent users in the system is very high. Therefore the utilization and fluctuations of resources especially for CPU are high, which makes it hard to get a well separated pattern of anomalies. The changes in the number of users are very low, so it simulates a workload without any recognizable trend.

Dataset3: We start to increase the concurrent number of users from a low level to a medium level which creates a visible trend in the number of users as well as resource utilization. The increase is performed by adding 10 users every 10 steps. However, due to medium level of resource utilization, anomalies still show distinctive patterns in the attribute space.

Dataset4: We start to increase the number of concurrent users from a low level to a very high level. Therefore it simulates a fast-changing workload sent to the web server and causes higher utilization and fluctuations compared to dataset3. The increase is performed by adding about 10 users every 7 steps.

Dataset5: We start to increase the concurrent number of users from a low level to medium level by adding 10 users every 5 steps. Due to the high rate of increase in request numbers, the noise from high fluctuations is affecting all parts of the data.

CloudStone helps to generate these workloads with dynamic characteristics of web loads, considering various patterns in terms of seasonality and trends in a stream of requests. Therefore, training and testing can be done on consecutive windows of time series as described in Section 3.5.

3.4.2 Data Preparation

When applying IForest to learn from training data with the injected anomalies, we found that a combination of multiple data transformations is required to improve the detection efficiency of algorithms. First, all the features with constant variance are removed as they usually do not provide new information about changes in the system and their

Algorithm 1: Data Preprocessing

input : $D = (X_1, X_2, ..., X_m)$, $X_i \in R^{n \times 1}$: D is a matrix of n records, each record including m features
Parameter: k: Moving Average Window Size
w: Piecewise Median Window Size
output : s: Normalized Extended Data

$s \leftarrow \emptyset$
for $X \in D$ **do**
  Extract seasonal S_x component using STL method from X
  Extract trend T_x component using Piecewise Median from X
  $R_x \leftarrow X - S_x - T_x$
  $s \leftarrow s \cup R_x$
end
for $X \in D$ **do**
  Normalize X
  `// Compute K-moving average`
  initialize all indicators in a_j to 0
  for $j \leftarrow 1$ **to** n **do**
    $a[j] \leftarrow Average(X[max(1, j-k) : max(1, j-1)])$
  end
  $s \leftarrow s \cup a$
end
return (s)

existence just increases the dimensionality of the problem. Second, different features have different ranges of values. For example, CPU values can be between 0 and 100, but memory can vary from 0 to 8192. Therefore, we apply a standard normalization to convert all the values to a range between 0 to 1.

Another point to mention is that collected datasets include values from different features over a period of time that create a time series of each feature. Existing trends and seasonality characteristics related to these time series change the pattern of normal data over time. Therefore, as Algorithm 1 shows, we have used STL (Seasonal and Trend decomposition using Loess) technique [140] which is a filtering procedure based on LOESS (local polynomial regression fitting) smoothing to decompose series of various features and extract the trend and seasonality components. However, as [141] shows, the trend component obtained with this method has a problem of introducing some artificial anomalies in the remainder of data which consequently affects the accuracy of anomaly detection algorithms. Therefore, we obtain the Piecewise Median introduced by [141] to calculate approximate trend of time series. The median has shown to be a more robust metric in the presence of anomalies compared to the average values. Having the seasonality and trend components, the remainder component of time series is calculated and have been used as extra features for each dataset.

3.4.3 Feature Smoothing and Time Dependent Information

The collected datasets have similar characteristics to time series that present the patterns in data through underlying trend and seasonality features. According to our observations, there are some transient spikes and noises which are introduced during runs of the benchmark. The noise can be a result of wrong or missing measurements or caused by specific characteristics of the benchmark at the start of each run. In order to reduce the effects of transient values, one can consider an average of the consecutive values in predefined time windows which help to smooth these variations. On the other hand, the time of occurrence for each measurement is an important feature which can affect the interpretation of the state of the system. In other words, the behavior of the system presented by nearby instances can affect the decisions to identify an instance as an anomaly

or not. The average value of the recently observed instances can help to have an understanding of the previous state of the system and better highlight significant changes between adjacent time windows. Therefore, to further improve our model and include the basic knowledge of time dependent changes in the dataset, we can extend the raw data with a summary of historical data as presented in lines 10-14 of Algorithm 1. To achieve this goal, a window of k previous samples for each instance is considered and the values of the attributes are averaged out, including them as new features for each sample. The technique which is known as k-point moving average helps to decrease the effect of transient spikes and adds time dependent information into the attribute space.

3.4.4 Anomaly Detection Approach

Upon receiving enough observations from data preparation module, IFAD is able to start the anomaly detection process. This process can be divided into two main parts: model generation based on the training data and anomaly identification for the test data. In the following, we explain these two parts in more details.

IFAD leverages Isolation technique as the core part of its functionality. This technique is a decision-tree based ensemble approach named Isolation Forest (IForest) introduced in [106, 107]. We choose Isolation technique for our anomaly detection problem based on our observations that the target types of anomaly in the cloud performance data usually change the values of metrics suddenly and these changes are rare compared to the normal behavior of the system. Therefore, we suggest that there is a high chance that these rare unseen values can be detected based on the partitioning of attribute space in the presence of normal points. This will eliminate the need for calculating distance or density which results in high memory and computation complexities. Traditional classification methods cannot deal with highly skewed distributions. Anomaly detection is a classic example of highly skewed data. Isolation based technique is extremely fast and has been shown to work with a wide variety of distributions of data and does not require prior knowledge of these distributions. They can also be run in parallel to detect anomalies and its cost is negligible compared to many existing anomaly detection algorithms. The basic assumption of IForest is that anomalies are rare and different and

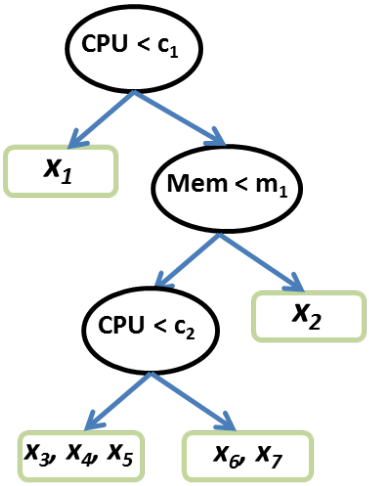


Figure 3.3: A simple Isolation Tree for two attributes CPU and Memory.

as a result, anomaly instances can be isolated faster than normal ones in the attribute space. This problem is formulated as a binary tree and each node is created by blindly (no need for the labelled dataset) selecting features and values as the conditions to split existing instances. The input of IForest for the model generation part is a sequence of n observations with extended attributes prepared as described in sections 3.4.1 and 3.4.2 to be used as the training data. In order to generate the first binary tree, IForest randomly selects $\psi <= n$ instances from the input observations. An attribute c from the column space of the training data and a value for the attribute is selected. Then, all ψ instances are divided into two categories based on the comparison of their values for the attribute c. The generated categories are assigned to two new nodes that create left and right children for the root node of the tree. The process of selecting an attribute and dividing existing instances into sub nodes repeats for new children nodes until the termination conditions are met, which means that there is only one instance left at the node or all the instances have the same values or the maximum length for the tree has been reached. Figure 3.3 shows a simple Isolation Tree for two attributes CPU and memory. Let $X = (x_1, x_2, ..., x_7)$ be a subsample of input observations to be isolated by their two columns. The root node divides X by selecting value c_1 for attribute CPU. As you can see in the figure, all instances except x_1 are moved to the right child node and x_1, as the only instance left, creates a leaf node at the left of the tree. The right child node divides

remaining instances by selecting value m_1 for memory which creates a leaf node at the right containing x_2 instance while other instances move to the left child node. This process continues until the conditions for termination are met. As we can see, instance x_1 can be a possible anomaly point in our sample set as it seems to be in a different range of CPU values compared to the other instances. To create an ensemble of the binary trees, this process repeats to generate t trees. The ensemble represents an abstract model of the current state of the system that can be referenced to find behavior deviations from the past.

The second part of the problem is the identification of the anomalies in the test data. Every test observation should traverse all the generated trees based on its values for the selected attribute of each node until it reaches a leaf node. The path length of the tree from the root to the leaf node represents the number of required partitions to isolate an instance on its values. The anomaly score is calculated based on the average path length of traversing the trees using a formula presented by [107].

We should highlight two points regarding the adaptability of Isolation technique in our target performance analysis problem. First, this method is an unsupervised learning approach which shows a *low linear-time complexity* with *small memory requirements*. These are essential characteristics for the problem of performance management in clouds to have fast and low-overhead solutions with the capability of finding previously unseen performance problems. Indeed, the worst time for training and testing of the algorithm is $O(t\psi^2)$ and $O(Lt\psi)$ respectively, where ψ is the number of selected subsamples and L is the size of testing dataset. This also leads to the conclusion that the training complexity is constant when the subsample size and the number of trees in the ensemble are fixed [107]. Furthermore, for the problem of anomaly detection in highly dynamic environments, there is a significant issue that usually is neglected about the impact of the workload heterogeneity in the accuracy of the models. The heterogeneity in web applications due to the resource configurations or internal and external events can change the normal pattern of data in the system. A fast anomaly detection procedure which does not require time consuming parameter tunings is an essential requirement which is satisfied by the IFAD framework.

3.4.5 Evaluation Metrics

In order to evaluate the detection accuracy of different algorithms, we distinguish four cases. True Positive (TP) that represents a case that an anomaly instance is reported correctly by the algorithm as anomaly. False Negative (FN) that shows a missed anomaly instance which is not detected by our algorithm. False Positive (FP) which refers to cases that normal instances are detected as anomaly and True Negative (TN) that shows a normal instance is correctly identified as normal. Considering these definitions, two metrics, True Positive Rate (TPR) and False Positive Rate (FPR) can be calculated based on Equation 3.1 and Equation 3.2. In probability based detection algorithms that calculate anomaly scores for each instance, the values TPR and FPR depend on the selection of a threshold that distinguishes anomaly instances from normal ones. Receiver Operating Characteristics (ROC) is a curve that represents a trade-off between TPR and FPR for different thresholds (cutoffs) of anomaly scores. Most of the existing works report the Area Under the Curve (AUC) for this curve as a measure of detection capability of an algorithm for target datasets.

$$TPR = \frac{TP}{TP + FN} \tag{3.1}$$

$$FPR = \frac{FP}{FP + TN} \tag{3.2}$$

However, in the addressed problem to identify anomalies in the cloud environment, the number of normal instances is much larger than anomaly instances which means that the positive and negative class labels are highly unbalanced. In the cases with highly unbalanced values for class labels, [142] shows that the RPAUC value may capture some patterns in the detection efficiency of algorithms that cannot be represented by ROC curve. PRAUC is calculated as the area under the curve for Precision and Recall values which can be calculated based on Equation 3.3 and Equation 3.4. Precision is a measure of the fraction of detected anomalies that are true anomalies and Recall is a measure of the fraction of true anomalies that can be detected by the algorithm.

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

These metrics combined together, enable us to have a better understanding of real capabilities of different algorithms for evaluation.

3.5 Performance Evaluation

The experiments are performed by deploying a realistic web serving benchmark on the Australian research cloud environment. In order to select a proper benchmark, some of the existing benchmarks such as RUBIS and PetStore that are extensively used in the literature to monitor the performance of VMs were considered [26, 63] . However, these benchmarks cannot capture the interactive functionalities of today's Web 2.0 applications. Therefore, for the implementation part, a 3-tier web application based on CloudStone benchmark is deployed [139]. CloudStone is part of the CloudSuite which is a benchmark for cloud services. The benchmark highlights the distinctive characteristics of Web 2.0 workloads and aims to generate real web workloads to capture web functionality in a scalable cloud environment. The three main layers of the benchmark are shown in Figure 3.4. It includes a Markov-based workload generator for emulating user requests, application and database servers. Workload generator enables the benchmark to have a fine-grained control over parameters that characterize the workload behavior [139]. CloudStone employs Faban, deployed on all the machines, to control the runs and emulate the user behavior. Application servers host a PHP based social network application in nginx servers. The generated requests sent from Faban client, are processed in application and database servers and the results are sent back to the client machine.

Benchmark represents a Web 2.0 social event application that mimics real user behavior in an interactive social environment with a combination of individual and social operations (such as creating events, tagging, attending an event or adding comments). Each request from the user includes a sequence of HTTP interactions between client and

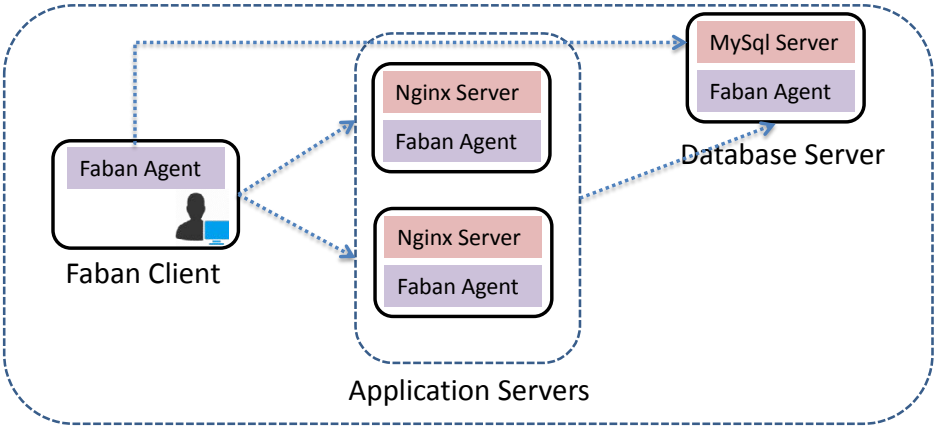


Figure 3.4: CloudStone Components.

server which accomplishes one of the mentioned tasks. We have also installed Ganglia that is a scalable, distributed monitoring component to monitor and collect performance indicators of system and applications.

3.5.1 Data Generation and Anomaly Injection

For the purpose of evaluating the performance of IFAD on workloads with different characteristics, five datasets are collected based on the specifications defined in Section 3.4.1 and each dataset includes about 15 hours of performance data. In order to generate each dataset, the workload generator starts to send a sequence of requests to the web server as part of the normal behavior of the system which generates time-series of performance and utilization data resulted from the interactions of user with the application. Then, anomalies are injected at random times to VMs hosting application or database server. The duration of different types of the anomaly may differ, but the contamination rate of the final data with anomaly instances is kept in the range 7-11% for all experiments. This is a rate that corresponds to a low anomaly intensity which is more common in cloud environment [77]. In the following experiments, five types of anomalies are tested:

Memory Load: A process is started on the same VM hosting the application server that allocates the available memory of the VM, but forgets to release it. As a result, after

Table 3.3: Experiment Configurations

Variable	Description	Value
Anomaly Contamination	The rate of anomaly instances in each dataset	$7\% - 11\%$
t	Number of trees	100
ψ	Number of random samples selected from each dataset	256
n	Total number of instances in training dataset	1650 - 1850

some time, the web application server encounters the problem of finding the required memory to process requests received as part of the normal operation of the system.

CPU Load: A CPU-intensive process is started on the VM that hosts the application server.

Disk Load: A series of I/O intensive tasks (read and write multiple files) are performed on the VM that hosts the database server.

Server Fault: The application server is shut down for some time. As there is no server available to respond to the requests, the utilization of the host VM decreases without any significant change in the number of incoming requests.

Flash Crowd: The number of concurrent users is suddenly increased to simulate a spike in the number of requests. Therefore, all measurements show a higher utilization of VM resources in response to this change.

The anomaly injection scripts are generated with the help of a variety of the packages including stress-ng and Cpulimit and the generator files for different types of anomalies are distributed to the target machines to be called by the master node at identified start times.

The final dataset, after applying preparation filters and adding new features, includes 29 features for the application server and 69 features for the database server. The dataset includes various types of performance indicators such as CPU and memory. Table 3.1 shows some of the major attributes collected for this dataset.

3.5.2 IFAD Settings

The base functionality of IFAD is on the assumption that anomalies are rare and different. To achieve this goal, IForest builds an ensemble of trees on a selected sample of data. The anomaly points are detected as instances with the shortest average path length on the generated trees. Corresponding to each node of the tree, one attribute is selected and the existing instances at the node are partitioned, creating two nodes based on the values of the selected attribute. In this work, we apply two different approaches for attribute selection phase of this algorithm:

- Random IForest (**IForestR**): This is the default procedure for attribute selection which splits each node of the tree based on a randomly selected attribute and a random value for this attribute.

- Deterministic IForest (**IForestD**) which tries to select an attribute that best divides the sample space into two categories with different distributions.

We develop and test anomaly detection models in IFAD using IsolationForest package implemented in R environment. The training parameters used in all the evaluations are the same and equal to the values presented in Table 3.3.

3.5.3 Evaluation Results

To validate system anomaly detection for web applications, we have conducted extensive experiments to evaluate different aspects of IFAD using several datasets collected from the deployed environment on Nectar virtualised environment. To perform comparisons, three unsupervised algorithms are also implemented as follows:

- KNN: The k-nearest neighbor distance is computed for each sample as a score of anomaly. The curve is computed based on the adjustment of cutoff value on the distance measure. In order to select k, we have tested different values from 2 to 10 and based on the results, a proper value is selected.

https://sourceforge.net/projects/iforest/

Table 3.4: AUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Dataset1	**90.3**	**95.0**	**95.2**	**94.8**	**92.4**
Dataset2	79.0	**83.2**	80.3	78.1	**87.0**
Dataset3	**92.2**	**92.0**	**91.3**	**91.5**	**88.9**
Dataset4	**88.3**	71.9	65.9	68.8	73.8
Dataset5	86.1	**92.0**	86.1	**88.5**	**89.6**

- One-class SVM (OCSVM): OCSVM is another algorithm with a non-linear kernel which calculates a soft boundary of normal instances and identifies outliers as data points that do not belong to the normal set. OCSVM is basically used for the novelty detection. However, as the selection of decision boundaries are soft, it can be applied in unsupervised problems as well. In order to select kernel parameter, we have tested different configurations through a 5-fold cross validation and selected the parameter γ based on the best results.

- L2SH: L2SH is a family of Locality-Sensitive Hashing isolation forests (LSHiForest) proposed by [143]. LSHiForest is a generalized version of the isolation based anomaly detection forests in which IForest and L2SH are two special cases of this family applying different types of the similarity measure and LSH functions. The base idea of LSH functions is that similar instances should be hashed to the same bucket with a higher probability than other non-similar instances. Therefore, in LSH trees, an internal node can be partitioned into more than two branches which is dependent on the number of buckets from hashing procedure. Regarding similarity measures, L2SH is associated with l2-norm or Euclidean distance.

Algorithms with random characteristics including IForestD, IForestR and L2SH are repeated 10 times in each experiment and the average of the results are reported. Though our methods are unsupervised, to be able to validate the accuracy of algorithms, we track the time of the anomaly injection and consider the indices of corresponding mea-

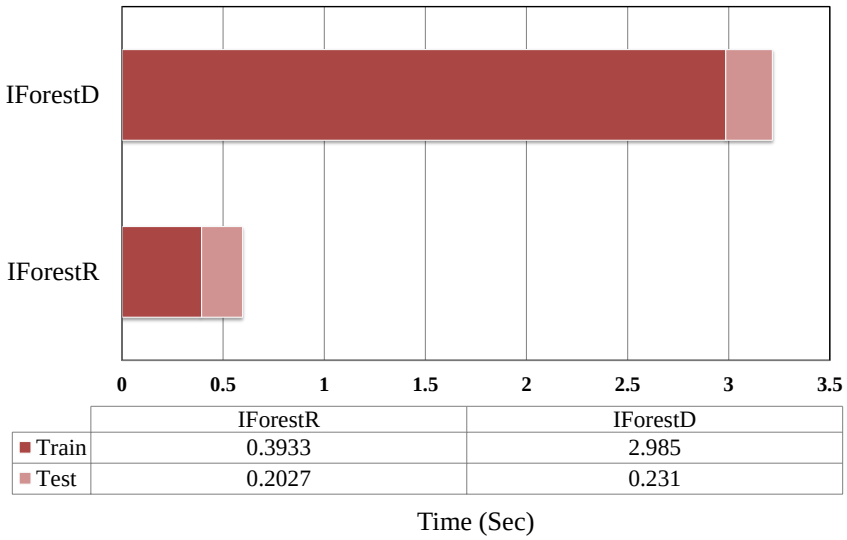
Table 3.5: PRAUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Dataset1	**75.9**	57.0	68.8	66.3	54.5
Dataset2	**47.0**	**44.0**	36.8	36.9	**45.1**
Dataset3	**67.6**	39.5	44.8	43.8	40.8
Dataset4	**54.7**	44.4	35.8	35.8	40.7
Dataset5	**50.0**	**53.6**	**47.3**	**50.8**	**49.2**

surements as the true anomaly points and the remaining measurements as true normal points.

For the first scenario, we train algorithms with both normal and abnormal instances. Then, each model is tested on another part of the data that includes all types of anomalies. The results for both metrics AUC and PRAUC are reported for all the datasets in Table 3.4 and Table 3.5. For each dataset, the best results with a difference of maximum 5 percent are highlighted. As the result shows, IForest can detect anomalies with high accuracy and performs particularly well in PRAUC with the highest results in all datasets.

KNN and IForestR perform well in 4 out of 5 datasets followed by IForesD and L2SH in terms of the AUC while IForestD also achieves high PRAUC for all the datasets. These observations are expected as IForestD tries to select the best splitting attribute at each node so there is a higher probability to isolate anomaly points with a very different distribution than normal points at the top of the trees while it may miss some points with more similar distributions to the normal space. For dataset1 and dataset3 other algorithms also show good AUC while their PRAUC is less than IForestD. Regarding dataset2 which shows a high variance in the values of collected attributes, anomaly instances can hardly be detected and as a result, the average precisions of all algorithms are low. Regarding other datasets, as the variance of data increases, data becomes more scattered and the pattern of anomalies can be masked by some of the normal instances resulting from high fluctuations in the data. In all these cases, IForestD shows good AUC and PRAUC compared to other algorithms.

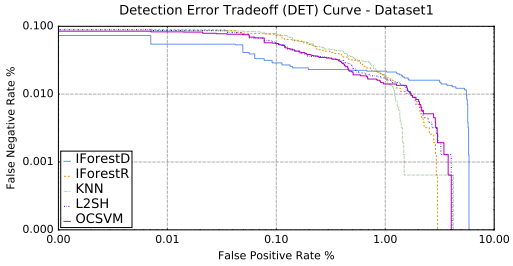


	IForestR	IForestD
Train	0.3933	2.985
Test	0.2027	0.231

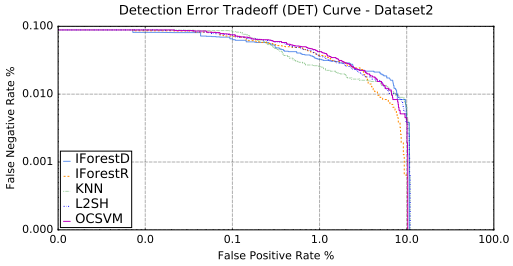
Figure 3.5: A comparison of train and test times for IForestR and IForestD. The average testing time for one instance is around 0.1 milliseconds considering the size of test datasets for different workloads.

Figure 3.5 shows a comparison of average training and testing times on all datasets between two versions of IForest used in this work to better demonstrate the effect of attribute selection complexity on the timing of the IForest. We observe that IForestR which selects the attributes randomly is very fast with a training time less than 1 second while IForestD has a training time around 3 seconds which is slower than the random version. However, in the worst case, updating of the models happens at each monitoring interval which is 15 seconds in our work and this interval can be higher based on the stability of application and environment [77]. Moreover, considering the number of instances in the test data, testing for one instance takes around 0.1 milliseconds. This is a reasonable result especially considering that the booting of new VM instances can take around 2 minutes or more based on the performance study done by [119].

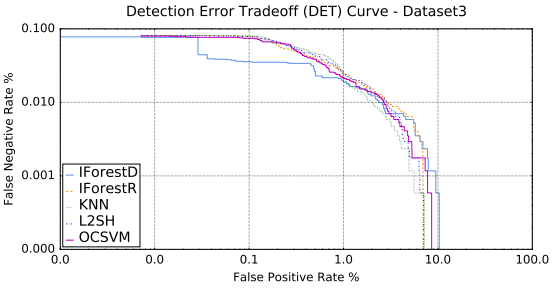
Figure 3.6 depicts the detection error trade-off (DET) curves for all algorithms and for each dataset. The curves are computed based on defining different thresholds on anomaly scores and computing the log rate of the missed anomalies (FN) and false alerts (FP). FN represents the rate of missing anomaly cases while FP is a measure of



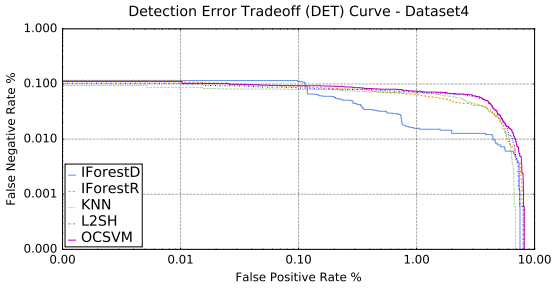
(a) DET Curves - Dataset1



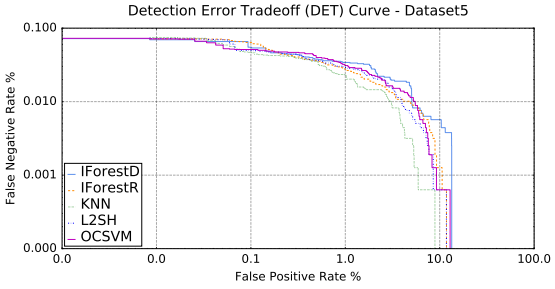
(b) DET Curves - Dataset2



(c) DET Curves - Dataset3



(d) DET Curves - Dataset4



(e) DET Curves - Dataset5

Figure 3.6: Plots of Detection Error Trade-off (DET) curves for all algorithms and different datasets

Table 3.6: Anomaly Detection for each type - AUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Memory	88.2	**94.0**	**93.0**	82.4	75.8
Disk	**97.5**	90.7	**95.9**	**96.4**	**96.1**
CPU	**99.4**	**96.0**	**96.7**	**98.44**	90.4
Load	88.2	**96.8**	**99.8**	**99.2**	**99.9**
Server	**96.4**	90.2	**92.3**	**93.4**	**93.9**

false alarms that can wrongly cause the application administrators to start preventive actions. This trade-off is an important observation especially for the applications that have tight restrictions on the accepted rate of positive/negative false alerts [130]. As we can see, no algorithm shows the best FP and FN for all thresholds or datasets which is expected especially due to the heterogeneity of datasets. For example, IForestD performs better in dataset1, dataset2 or dataset3 for FP less than 1%. The observed results confirm the idea that we need to have a more precise understanding of the real requirements of application to be able to select proper approaches that fit the specifications of our problem. This can be achieved by manually identifying the preference of the applications in terms of the precision or recall values or using the concepts of majority voting and ensemble approaches which try to combine the results of several algorithms. As an example case, for prevention mechanisms that target disk related problems with expensive mitigation actions, one may prefer to have a method that has a high precision with minimum false alarms. In contrast, for the Load problems, the high detection rate of the problem may be more important, so an algorithm with a better recall value is preferred.

For the second set of experiments, we investigate the detection performance of different methods for each type of anomaly. The results are shown in Table 3.6 and Table 3.7. All methods have a high AUC value for CPU anomaly which shows they can accurately identify anomaly points corresponding to the high utilization of CPU. However, IForestD also shows a high precision for this type of anomaly that represents a lower false alarm rate. For Disk and Server anomalies, IForesetD again shows a high AUC and

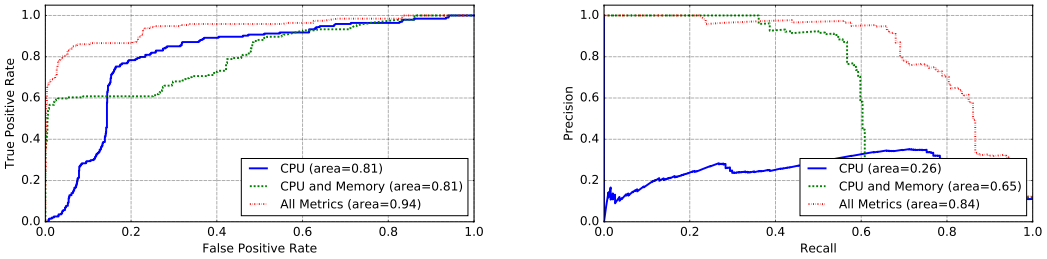
Table 3.7: Anomaly Detection for each type - PRAUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Memory	44.1	68.9	**89.2**	64.8	50.9
Disk	**95.1**	32.8	67.0	84.0	78.8
CPU	**93.9**	75.4	79.7	86.0	59.5
Load	44.1	68.6	**98.7**	91.0	**99.5**
Server	**76.4**	46.1	58.3	67.2	69.4

RPAUC value compared to other algorithms. However, it has a low precision for memory and Load anomalies. The reason can be due to the gradual increase of these two types of anomalies that create a denser cluster of anomaly points which can decrease the difference between normal and abnormal anomaly scores. L2SH has a more stable performance in these cases and usually avoids the worst case performance in different scenarios.

Finally, we show the effect of multi-attribute compared to the single attribute performance analysis. We have repeated experiments with the IForestD algorithm for three scenarios of feature selection. In the first run, we include the CPU metric as the only feature to detect anomalies. In the second run, we add the Memory feature and obtain the result of anomaly detection based on a combination of two features. We compare these two scenarios with another run of the algorithm on all collected features. The comparison is performed by measuring AUC and PRAUC metrics and shown in Figure 3.7. We can see that the single metric of CPU is not that much informative as we miss many anomaly points and the precision of detection is very low. When we consider both features of CPU and Memory, the results of AUC and PRAUC show significant improvements. However, including all the metrics shows further improvements in the results of the anomaly detection. This leads us to conclude that in a dynamic environment with different types of anomalous problems, a combination of multiple metrics is much more informative and precise than single-feature based solutions.

In summary, the proposed IFAD framework shows higher levels of precision for a



(a) ROC Curves **(b)** RPROC Curves

Figure 3.7: Plots of ROC and PRROC for IForestD algorithm based on different metrics

range of datasets and anomalies. These results accompanying the unsupervised fast execution of anomaly detection process and the ability to work with the default configurations in various types of workloads which reduces the overhead of tuning steps during model updating makes it a good candidate for applications with a highly dynamic nature, demanding higher precisions or requiring to perform in completely unsupervised manner.

3.5.4 Time Complexity

In order to have a better understanding of the performance of proposed method, we identify the main blocks of preprocessing and behavior learning steps as follows: The main parts of any anomaly detection framework are data preparation and model generation/testing. Algorithms 1 shows the detailed steps of data preparation based on the concepts of time series analysis. The input is a matrix of n rows (instances) with m columns (features). Assuming fixed seasonality patterns with default parameters, the complexity of data preparation step which is dominated by STL process is O(mn).

Considering that all target models in our work can take the advantage of detrending and seasonality smoothing done in the preparation phase, the main difference in the runtime complexity of the evaluated learning algorithms comes from model generation and parameter tuning. As explained in [107] and section 3.4.4, considering a constant number of trees and sub sampling size for each isolation tree, the training and space complexities of IForest are constant which makes it suitable for large datasets. L2SH is

another version of Isolation-Tree based methods that utilizes locality sensitive hashing. While the distance measure is different compared to IForest version, it shows the same runtime complexity[143]. In contrast, OCSVM and KNN both need the pre-tuning of parameters and show higher runtime complexities. OCSVM involves a quadratic programming problem which increases the complexity between O(n^2) to O(mn^2) depending on the cashing capabilities and the sparsity of columns. The KNN algorithm requires the computation of the distance to recognize anomaly points. The referenced K for distance calculations highly depends on the distribution of data and one needs a careful testing of different K values especially when the workload characteristics change over time. Having an efficient data structure for implementation, the complexity of the algorithm can be improved to O($mlogn$). Referring to the comprehensive analysis of Isolation-Tree based methods in [107] which shows the robustness of the algorithm with the default values of parameters (100 trees, 256 sample size) and the possibility of having a parallel implementation for ensemble trees generation to further improve the speed, Isolatin-Tree based anomaly detection shows a promising capability for environments where the models need to be updated regularly.

3.6 Summary

This chapter presents IFAD framework which utilizes the concept of Isolation-Trees to detect abnormal behavior in the time series of performance data collected from the application and underlying resources. In addition, the effects of different performance anomalies on various types of the workload in a web-based environment are investigated. The results show that IFAD achieves good AUC and higher precision in detecting performance anomalies. Another observation highlights that depending on the type of heterogeneity in the workloads or changes in the performance of resources, some algorithms can have a better detection rate or average precision. Moreover, a combination of different metrics can improve the learning process compared to single metric solutions based on the common features CPU or Memory. IFAD can be utilized as the anomaly detection module in a resource auto-scaling framework where the knowledge from detection process can help to recognize the possible anomalies in the system behavior.

Our method addresses the problem of resource bottleneck identification in the web-based application where the target anomalous behavior is due to the large changes of attribute values. The fast and memory-efficient execution of IFAD makes it a good approach for detecting anomalies in fast changing environment. However, another problem for on-time processing of high-volume information is dealing with datasets with many attributes. Therefore, in the next chapter, we propose a feature refinement process to improve the efficiency of anomaly detection process with regard to high-dimensional datasets.

Chapter 4

An Isolation-Tree based Learning of Features for Anomaly Detection

Isolation-based method is an effective approach for detecting anomalies. However, a common challenge of iTrees as well as other anomaly detection techniques is dealing with high dimensional data potentially consisting of many irrelevant and noisy features. This is an important issue for cloud hosted applications where a variety of problems can affect different groups of features. Therefore, refining the feature space for removal of irrelevant attributes is a critical issue. In this chapter, we introduce an iterative iTree based Learning (ITL) algorithm to handle high dimensional data. The results show that ITL can achieve significant speedups with appropriate choice of the number of iTrees while achieving or exceeding AUC values of other state of the art Isolation-based anomaly detection methods.

4.1 Introduction

Anomaly detection is an important field of the knowledge discovery with a rapid adoption in a variety of applications. In the context of cloud environment, this process is utilized for a variety of performance management applications. For example, intrusion detection systems provide frameworks that monitor the performance of the network to find misbehaving users, possible misconfiguration or serious conditions from an attack on the system [19]. Similarly, SMART(Self-Monitoring, Analysis and Reporting Technology)-based disk failure prediction applications perform regular monitoring and

This chapter is derived from:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ITL: An Isolation-Tree based Learning of Features for Anomaly Detection in Networked Systems, *Future Generation Computer Systems (FGCS)*(under 2nd review).

anomaly detection analysis to increase the reliability of storage systems [144].

With the advances in data collection techniques, storage capabilities and high performance computing, a huge volume of monitoring data are collected from continuous monitoring of the system attributes. Despite the appealing benefits of access to larger amounts of the data for better diagnostics of the anomalous events, the great challenge is how to deal with high volume of information that should be processed effectively in real-time. The increase in the volume of data is due to: 1) Recording of fine grained measurements for long periods of time which increases the number of records to be processed. 2) High dimensional data with many features that describe various aspects of target system. The curse of dimensionality or having many features can make the problem of anomaly detection in high dimensional data more complex in terms of the runtime efficiency and accuracy [145]. This is also becoming a critical issue in cloud systems which are exposed to several performance problem at different layers of computing. As a result, the collected performance data is heterogeneous and includes a variety of attributes from low-level operating system logging data to hardware specific features, applications performance data or network related information. On the other hand, these performance data are exposed to a variety of problems such as different types of attack and intrusion patterns in network related performance data. Particularity, the general anomaly detection techniques can not perform well for high dimensional network data with a variety of data types and embedded meaningful subspaces [146]. Moreover, the collected data is dynamic and rapidly changing. All of these, together, highlight the need for highly adaptable and fast analytic solutions. Therefore, researchers are investigating more efficient techniques with the goal of better explorations of collected data and improving the quality of the extracted knowledge.

Traditional anomaly detection algorithms usually work based on the assumptions that highly deviated objects in terms of the common metrics such as distance or density measures have a higher probability of being anomalous. While these assumptions are applicable in general, their accuracy can be affected when the base assumptions do not hold, such as in the high dimensional data [147, 148]. Moreover, in the traditional methods, anomalies are detected as a by-product of other goals such as classification and clustering. More recent approaches, such as isolation-based methods directly target the

problem of anomaly detection with the assumption that anomalies are few and different [107, 149]. However, the problem of having high number of noisy features can also affect these methods. In order to improve the efficiency of detection algorithms in high dimensional data, a variety of solutions such as random feature selection or subspace search methods are proposed [150, 151]. However, the proposed approaches are usually considered as the preprocessing steps which are performed as a separate process from the anomaly detection. Although this separation makes them applicable for a variety of algorithms, finding the relevant features in the datasets with many noisy features can be challenging when the mechanism of target detection algorithms in finding the anomalies is ignored. Therefore, a question arises that is there a way that one can improve the efficiency of anomaly detection by extracting knowledge from the assumptions and the process which leads to the identifying potential anomalies in the data? Having this question in mind, in this chapter, we address the problem of anomaly detection in high dimensional data by focusing on the information that can be extracted directly from the Isolation-based mechanism for identifying anomalies. The reason for selecting this technique as the base process is that it is known as a category of anomaly detection techniques that is designed to directly target the most common characteristics of anomalous events such as rarity compared to other objects. We exploit the knowledge that comes from the detection mechanism to *identify* the features that have higher contribution in the separation of the anomaly instances from normal ones. This approach helps to identify and remove many irrelevant noisy features in high dimensional data. The proposed method, Isolation-Tree (iTree) based Learning (ITL), addresses the problem of anomaly detection in high dimensional data by refining the set of features with the aim of improving the efficiency of the detection algorithm. These are the features that appear in the short branches of iTrees. The refining procedure helps the algorithm to focus more on the subset of features where the chances of finding anomalies are higher while reducing the effect of noisy features. The process helps to obtain more informative anomaly scores and generates a reduced set of the features that improves the detection capability with better runtime efficiency in comparison to the original method that uses all the features. The contributions of this work are therefore, an iterative mechanism for structural learning of data attributes and refining features to improve the detection efficiency

of Isolation-based methods and reduce the effect of noisy and irrelevant features. The simplified model is extremely fast to train so that the model can be periodically trained when the important features largely remain unchanged.

We have compared ITL with the state of the art feature learning based framework [108] and show that not only ITL improves the results as an ensemble learning method with the bagging of scores, but also it can discover a subset of the features that can detect anomalies with reduced complexity.

The remainder of this chapter is structured as follows. Section 4.2 reviews some of the related works in the literature. Section 4.3 overviews the main assumptions in the problem formulation. Section 4.4 presents ITL framework and details the steps of the algorithm. Section 4.5 presents experiments and results followed by time complexity and runtime analysis, and finally Section 4.6 summarizes the chapter findings.

4.2 Related work

The general concept of anomaly detection indicates the exploration and analysis of data with the aim of finding patterns that deviate from normal or expected behavior. The concept has been widely used and customized in a range of applications such as financial analysis, network analysis and intrusion detection, medical and public health, and etc [19, 131, 152]. The growing need for anomaly related analysis has led researchers to propose new ways of addressing the problem where they can target unique characteristics of the anomalous objects in the context of the target applications. For example, distance based algorithms address the problem of anomaly detection based on the distance of each instance from neighborhood objects. Greater the distance, the more likely that the instance presents abnormal characteristics in terms of the values of the features [132, 153]. Alternatively, [133, 134] define the local density as a measure for abnormality of the instances. The objects with a low density in their local regions have a higher probability of being detected as anomaly. Ensemble based methods try to combine multiple instances of anomaly detection algorithms in order to improve the searching capability and robustness of the individual solutions [108, 154].

Performance anomaly detection has also widely been applied in the context of cloud

resource management to identify and diagnose performance problems that affect the functionality of the system. These problems can happen at different levels of granularity from code-level bug problems to hardware faults and network intrusions. The fast detection of problem is a critical issue due to the high rate of changes and volume of information from different sources. A variety of techniques from statistical analysis to machine learning solutions are used to process collected data. For example, Principal Component Analysis (PCA) is used in [20] to identify most relevant components to various types of faults. [144] apply random forests on various exported attributes of drive reliability to identify disk failures. [27] exploits self-organizing map technique to proactively distinguish anomalous events in virtualized systems. Clustering techniques are utilized by [22] to split the network related log data into distinctive categories. The generated clusters are then analyzed separately by anomaly detection systems to identify intrusion and attack events. [102] uses entropy concept on network and resource consumption data to identify denial of service attacks.

While the above mentioned approaches show promising results for a variety of problems, the exploding volume and speed of the data to be analyzed require complex computations which are not timely efficient. A common problem which makes these difficulties even more challenging is the high dimensional data. For example, the notion of distance among objects loses its usability as a discrimination measure as the dimension of data increases [145, 147]. Methods based on the subspace search or feature space projections are among approaches which are proposed as possible solutions for these problems [155]. The idea of dividing a high-dimensional data to groups of smaller dimensions with related features is investigated in [156]. This approach requires a good knowledge of domain to define meaningful groups. PCA based methods try to overcome the problem by converting the original feature set to a smaller, uncorrelated set which also keeps as much of the variance information in data as possible [157]. PINN [158] is an outlier detection strategy based on the Local Outlier Factor (LOF) method which leverages random projections to reduce the dimensionality and improve the computational costs of LOF algorithm. Random selection of the features is used in [159] to produce different subspaces of the problem. The randomly generated sub problems are fed into multiple anomaly detection algorithms for assigning the anomaly scores.

While random selection can improve the speed of feature selection process, as the selection is completely random there is no guarantee of having informative subspaces of data to improve the final scoring. [150] and [108] propose two different variations of subspace searching. The former tries to find high contrast subspaces of the problem to improve the anomaly ranking of density based anomaly detection algorithms. The subspace searching is based on the statistical features of the attributes and is performed as a preprocessing step separated from target anomaly detection algorithms. The latter, in contrast, integrates the subspace searching as a sequential refinement and learning in anomaly detection procedure where the calculated scores are used as a signal for the selection of next subset of the features. Our proposed anomaly detection approach is inspired by such models and tries to refine the subset of the selected features at each iteration. However, we try to take advantage of the knowledge from the structure of constructed iTrees instead of building new models for the regression analysis.

4.3 Model Assumptions and an Overview on Isolation-based Anomaly Detection

The iterative steps in ITL process are based on the iTree structure for assigning the anomaly scores as well as identifying features. We choose isolation-based approach and specifically IForest algorithm in this work due to its simplicity and the fact that they target the inherent characteristic of the anomalies. We note that the target types of the anomaly in this research are instances which are anomalous in comparison to the rest of the data and not as a result of being part of the larger groups [19]. This is also consistent with the definition of anomaly in many cloud related performance problems especially network and resource consumption abnormalities.

The idea of Isolation-based methods is that for an anomaly object we can find a small subset of the features that their values are highly different compared to the normal instances and therefore it can quickly be isolated in the feature space of the problem. IForest algorithm demonstrates the concept of the isolation and partitioning of the feature space through the structure of trees (iTrees), where each node represents a randomly selected feature with a random value and existing instances create two new child nodes

based on their values for the selected feature. It is demonstrated that the anomaly instances usually create short branches of the tree and therefore, the *length of the branch* is used as a criterion for the ranking of the objects [106]. Consequently, anomaly scores are calculated as a function of the path length of the branches that isolates the instance in the leaf nodes on all generated iTrees. This process can be formulated as follows [107]: Let $h_t(x)$ be the path length of instance x on iTree t. Then, the average estimation of path length for a subset of N instances can be defined as Equation 4.1:

$$C(N) = \begin{cases} 2H(N-1) - 2\frac{(N-1)}{N} & if N > 2, \\ 1 & if N = 2, \\ 0 & otherwise \end{cases} \tag{4.1}$$

where $H(N)$ is the harmonic number and can be calculated as $ln(N) + Euler\_Constant$. Using $C(N)$ for the normalization of expected h(x) of instance x on all trees, the anomaly scores can be calculated as follows:

$$s(x, N) = 2^{-\frac{E(h(x))}{C(N)}} \tag{4.2}$$

Considering this formula, it is clear that anomaly scores have an inverse relation with the expected path length. Therefore, when the average path length of an instance is close to zero, the anomaly score is close to 1, and vise versa.

Figure 4.1 shows a graphical representation of the isolation technique for a dataset with two attributes X_1 and X_2. The left and right columns show examples of random partitions on the attribute space and their corresponding tree structures to isolate a normal and anomaly instance respectively. As it is shown, instance A (anomaly) can be isolated quickly considering the sparsity of values of X_1 around this instance. Though this example is a simple case with just two attributes, the general idea can be extended to the problems with many features and variety of distributions.

Considering the above explanations, ITL process is based on an idea that iTrees can also give information on important features for detection purposes. Therefore, ITL analyzes the generated iTree structure to extract information about the features that have more contribution in creating short branches and detected anomalies. In order to bet-

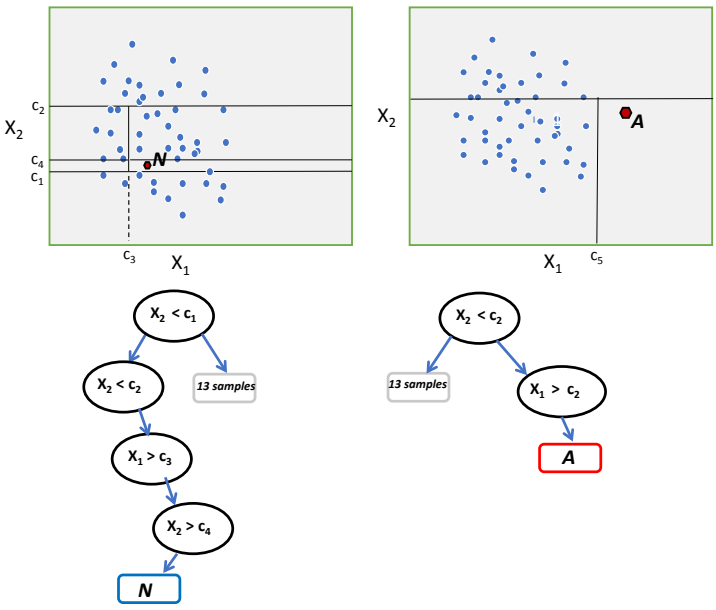


Figure 4.1: Isolation-based anomaly detection. iTree structures are used to represent the partitioning and isolation process of instances in a dataset with two attributes. The left and right columns show example sequences of partitions to isolate normal and anomaly instances, respectively.

ter explain the problem, let us assume that the input D is a matrix of N instances, each instance explained with a row of M features such that:

$$D = \{(X_i), 1 \leq i \leq N | X_i = (x_{ij}), 1 \leq j \leq M, \\ x_{ij} \in R\} \tag{4.3}$$

We have excluded nominal data in our assumptions and definition of Equation 4.3. However, the ITL process is general and can be combined with solutions which convert categorical data to numerical to cover both cases [108]. We formulate the problem as follows: Given matrix D as the input, we try to iteratively remove some irrelevant features from the feature space of D, keeping the more relevant features for the detected anomalies at each step in an unsupervised manner. The goal is to increase the quality of the scores in terms of assigning higher scores to the true anomaly points by reducing the effect of noisy features. The output at each step k is a set of the scores S_k on a set of

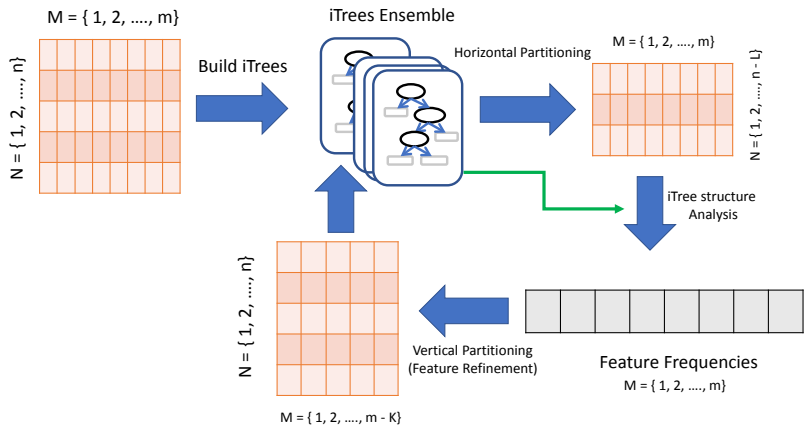


Figure 4.2: ITL Framework. The initial input is a matrix of N instances with M features. An ensemble of iTrees is created. Then, top ranked identified anomalies are filtered. The iTrees are analyzed for filtered instances to create a list of ranked features.

the reduced features M_k. The idea is that the removal of noisy features makes it easier to focus on the relevant partitions of the data, where the values of the features show higher deviations for the anomalous objects in comparison to the normal ones. As a result, the ranking of the input objects would improve with regard to the true detected anomalies.

4.4 ITL Approach

Figure 4.2 shows the main steps in ITL framework. As we already discussed in Section 4.3, the iTree structure forms the base of the ITL learning phase following the assumption that short branches in the structure of iTree are generated by the attributes with higher isolation capability. In another word, a subset of the attributes which are creating the nodes in the short length branches can form a vertical partition of the data that localize the process on anomaly related subset of the data. As we can see in Figure 4.2, the process is completely unsupervised with the input matrix as the only input of each iteration (that is we have no information of anomalous instances). There are four main steps in the ITL process and these are:

1. Building iTrees Ensemble: IForest creates a set of the iTrees from input data. This is a completely unsupervised process with random sampling of the instances/features

Algorithm 2: ITL Process

input : $D = (X_1, X_2, ..., X_N)$, $X_i \in R^M$: D is a matrix of N records, each record including M features

Parameter: *th*: Anomaly score threshold value

output : Reduced Matrix, Scores

1 $D' \leftarrow D$
2 **while** *not (There are unseen features)* **do**
3 Build *iTrees* ensemble using iForest on D'
/* Calculate scores for all input instances using Equation 4.2 */
4 $S = (S_k) = (s_{k1}, s_{k2}, ..., s_{kN}) \leftarrow Scores(iTrees, D')$
/* Filter a small part of the input matrix with higher anomaly scores */
5 $D\_subset \leftarrow \{x_i |\ x_i \in D' \ \&\& \ s_i \geq th\}$
6 initialize *Frequency* as an array with length equal to number of features in D_subset all equal to zero
7 **for** $tree \in iTrees$ **do**
8 **for** $x \in D\_subset$ **do**
9 update *Frequency* of features by adding the occurances of each attribute seen while traversing from root node to the leaf node that isolates x
10 **end**
11 **end**
12 $D' \leftarrow \{x_{ij} | x_{ij} \in D' \ \&\& \ frequency(j) \geq Average(frequency)\}$
13 **end**
14 return(D', S)

to create the splitting nodes in each tree.

2. Horizontal partitioning: The anomaly score for each instance is computed based on the length of the path traversed by the instance on the generated iTrees [107]. The final score shows the degree of outlierness for the instance. Our goal is to discover important features based on their contribution to isolation of anomaly instances. The low score instances do not affect the determination of the important features for anomaly detection and therefore, we can remove them reducing the data size.

3. Extracting Feature Frequencies: We create a frequency profile of occurrences of different features observed during traversal of short branches of iTrees. These features have high probability of detecting anomalous instances.

4. Vertical Partitioning: Having a profile of the feature frequencies, a subset of the features that are identified to have a higher contribution in the abnormality of data are selected and other features are removed. This process creates a vertically partitioned subset of data as the input for the next iteration of the ITL.

This process is repeated multiple times until the termination condition is met. As we continuously refine the features, we expect to see improvement in anomaly detection process as the detection process becomes more focused on the interesting set of features. Therefore, the set of iTrees built during consecutive steps can be combined to create a sequence of the ensembles. Algorithm 2 shows pseudo code of ITL. A more detailed and formal description of the process is presented in the following section.

4.4.1 Feature Refinement Process

We assume that the input D is a matrix of objects labeled as one of the classes of normal or anomaly. These labels are not part of the ITL process as it is an unsupervised mechanism. They are used for evaluating the output results of proposed algorithms and other benchmarks for validation purpose. The goal is to find a ranking of the objects, so that the higher values imply higher degrees of abnormality. Considering this objective, the first step of ITL process is to build the initial batch of the iTrees from input matrix.

IForest is used to create t iTrees. To create each iTree, ψ random instances are selected from D and each node of the tree is created by randomly selecting a feature and a value and splitting the instances based on this selection to form two branches. The output is iTrees ensemble and anomaly scores $S = (s_1, s_2, ..., s_N)$ computed for all instances based on Equation 4.2 (Lines 3-4, Algorithm 2).

After creating new iTrees, the next step is to reduce the target instances to be used for the learning procedure (Line 5). A threshold value (th) is defined and all instances with an anomaly score lower than this value are discarded. The idea behind this selection is to focus better on parts of the data which have higher degree of abnormality based on the iTrees structure as well as reducing the complexity of the problem. As the learning phase is the most time consuming part of the ITL process, this reduction dramatically decreases the runtime of the algorithm. The selection can also take advantage of the expert knowledge on the characteristics such as the contamination ratio of the dataset for defining a proper cut-off value of anomaly scores. The output of this step is a subset of the input matrix D (D') with p instances such that $p << |D|$. We emphasis that the process is unsupervised as we do not have the knowledge of anomalous instances. However, based on the assumption that anomalies are few and different, we expect to see many of the anomaly instances in D'. It should also be noted that the generation of each iTree is completely random in terms of the splitting features and value selection. Therefore, one tree may not be informative per se. However, when the random process is repeated to generate many numbers of trees, the overall observed patterns confirm the idea of short branch isolation of anomaly instances [107]. This can be observed in Figure 4.1 as well. Generating iTree structure on high density regions requires many nodes and splitting conditions to isolate one instance, while for an anomaly instance there is one feature or more that can quickly differentiate that from the rest of the data.

The instances that passed the filtering procedure from previous step (highly ranked anomalies) are processed by each iTree from ensemble model to record the frequency of occurrences of features when traversing the trees. The frequency profile of the features allows determining the important features relevant to detecting target anomalies. According to the formulas in Section 4.3 and their interpretation as an iTree structure, we expect to see a subset of more important features for anomalous instances in the short

branches of trees. It should be noted again that these are the expected observations from an ensemble of many random trees and are not attributed to any specific iTree. Consequently, we keep the features whose frequencies are higher than the average of the frequency profile (Lines 6-12).

The above steps are repeated multiple times. The output is a set of the anomaly scores for each subset of the data, starting from full data with all features. Therefore, the iteration k of ITL process creates a set S_k of anomaly scores for all instances on reduced feature set M_k (M_0 is the full set of the features for the first iteration). We note that each iteration would produce potentially different sets of anomalous points and hence different frequency profile of the features. The termination condition we choose is when the frequency of occurrences for all features is greater than one, meaning that every feature has seen at-least one anomalous point in the short branches of iTrees. The idea behind this condition is that as the noisy features are removed during the iterative process, ITL produces better iTrees for detecting true set of anomalies. Therefore, the observed features become more important in the detection process. When ITL reaches a state that all the features are present in the short branches, it indicates that all current features are contributing to the detection of anomalous instances. Therefore, the termination condition T_k at iteration k is evaluated as follows:

$$T_k = \begin{cases} True & \text{if } Size(M_k) \leq 1 \text{ or} \\ & \quad Frequency_k > 0 \\ False & \text{otherwise} \end{cases} \tag{4.4}$$

where $Size(M_k)$ evaluates the number of remaining features at the iteration k. $Frequency_k$ is the corresponding frequency profile which is an array of length M initialized by zero (Lines 6). The term $Frequency_k > 0$ evaluates the condition that the frequencies of all attributes in M_k are greater than zero. When T_k evaluates to true, ITL process terminates and the final outputs are evaluated as follows:

- *Bagging of the Scores*: Each iterative step of the ITL process produces score for each data point in D which represents the degree of anomalousness based on the corresponding set of the reduced features. As we try to improve the detection capability

of the ensemble by reducing the noisy features, we expect to get better anomaly scores in terms of the ranking of instances. Therefore, in this approach, the goal is to take advantage of the detection results from all iterations by averaging of the scores and defining a new score for each instance. Accordingly, the final score of each instance is calculated as follow:

$$S_f(x) = \frac{1}{K} \sum_{k=1}^{k=K} S_k(x) \tag{4.5}$$

where S_f is the final score and S_k is the score at iteration k from K iterations of ITL process.

- *Reduced Level Scores*: ITL produces an ensemble of iTrees on the important features for the anomaly detection. The generated iTrees on the reduced features can be used for detecting anomalies in new data. Therefore, the anomaly scores are calculated directly based on the extracted reduced feature set from the process.

4.5 Experiments

In this section, an empirical evaluation of ITL process on two network intrusion datasets and three benchmark datasets is presented. Two sets of the experiments are designed to show the behavior of ITL in bagging and reduced modes on the target datasets. First, Section 4.5.1 presents the datasets and parameter settings of the experiments. Then, Section 4.5.2 shows the comparison results of ITL in the bagging mode with a recently proposed state of the art sequential ensemble learning method and then investigates the improvements made by reduced level features in terms of both AUC and runtime analysis in a set of the cross validated experiments. Section 4.5.2 and Section 4.5.3 discuss runtime complexity and weakness/strength points of ITL approach.

Table 4.1: Properties of Data used for Experiments. *N* and *M* are number of instances and features in each dataset, respectively.

	N	M	Anomaly Ratio (%)
DOS	69363	37	3
U2R	69363	37	3
AD	3279	1558	13
Seizure	11500	178	20
SECOM	1567	590	6

4.5.1 Experimental Settings

Table 4.1 shows a summary of statistics for the benchmark datasets. All datasets are publicly available in UCI machine learning repository [160]. For U2R and DOS datasets which are network intrusion data set from Kddcup99, a down-sampling of attack classes is performed to create the anomaly class. In other datasets, the instances in minority class are considered as the anomaly.

In order to evaluate the results, we select Receiver Operating Characteristics (ROC) technique and present Area Under the Curve (AUC) as a measure of the accuracy of the system which summarizes the trade-off between true positive and false positive detection rates.

ITL process is implemented based on the publicly available python library, scikit-learn [161]. Unless otherwise specified, the values of the parameters for iTree generation step of ITL process are according to the recommended settings as explained in [107]. The values of other parameters are set based on the experimental tunings. The threshold value for the horizontal partitioning (*th* in Algorithm 2) is determined by assuming a contamination ratio equal to 0.05% for all datasets. This means that the cut-off threshold is identified so that 0.05% of the objects have a score higher than the *th* which is good enough considering the number of instances and the contamination ratio in our target datasets. The frequency profiling is done on the branches with maximum length of 4. To ensure comparability, the number of trees for the IForest algorithm in all methods is the

http://archive.ics.uci.edu/ml

Table 4.2: AUC results for the base IForest, ITL and CINFO. M and $M^{'}$ show the size of the original and reduced features for ITL. The best AUC for each dataset is highlighted in bold.

	IForest	CINFO	ITL	ITL Feature Reduction		
				M	$M^{'}$	Reduction
DOS	**0.981**	0.971	**0.981**	37	21	43%
U2R	0.874	0.894	**0.901**	37	18	51%
SECOM	0.551	**0.655**	0.594	590	80	86%
AD	0.704	0.850	**0.856**	1558	54	97%
Seizure	0.989	0.987	**0.990**	178	163	8%

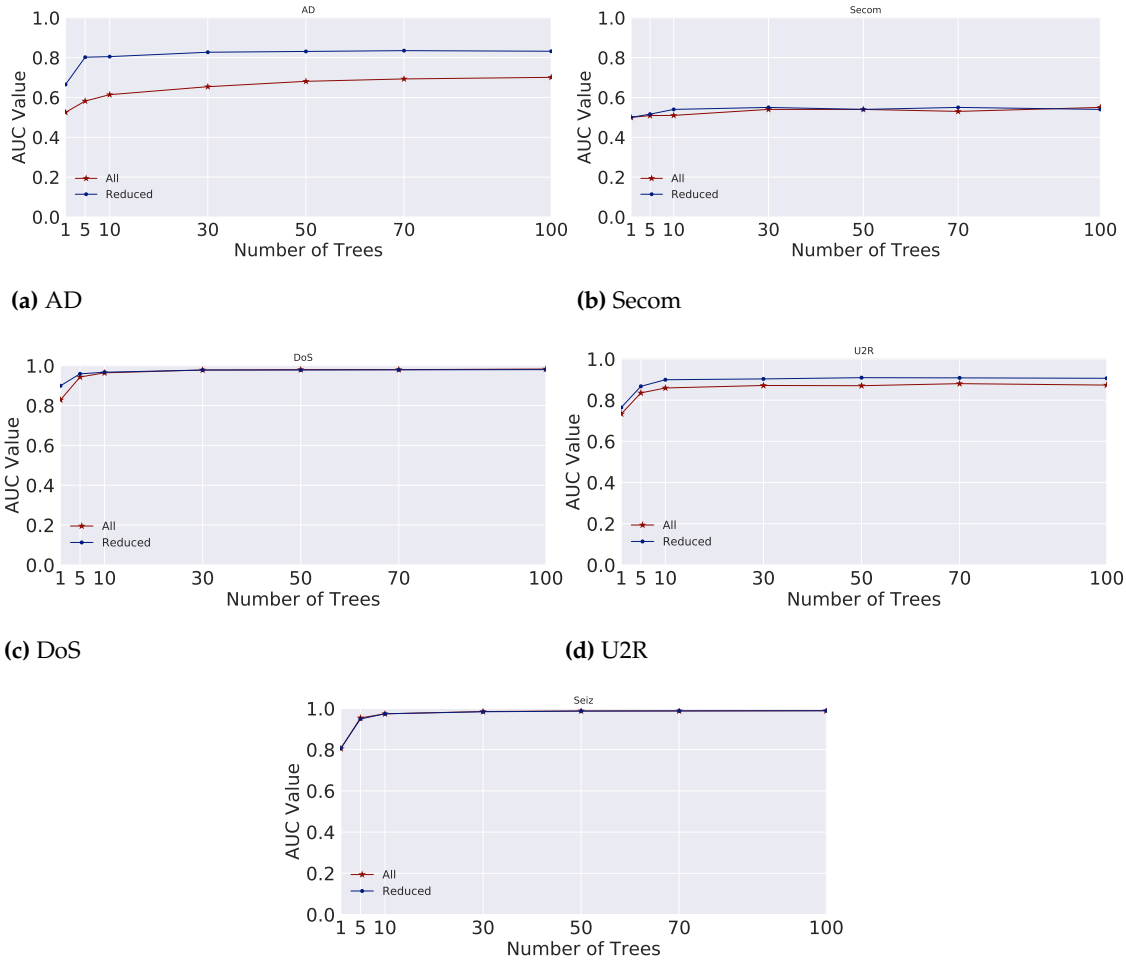
same and is between 600 to 900 trees.

For the comparison, we have selected a recently proposed sequential learning method, CINFO, designed for outlier detection in high dimensional data [108]. CINFO works based on lasso-based sparse regression modeling to iteratively refine the feature space. As their method is general, we select the IForest based implementation which considers the scores generated by IForest algorithm as the dependent variable of the regression model. Due to randomness feature of iTree generation, each experiment is repeated for minimum of 5 times and the averages of results are reported. For CINFO, the number of repeated experiments is based on their recommended values to have stable results [108].

4.5.2 Experiment Results

ITL with Bagging of the Scores

Table. 4.2 shows the AUC results for the base IForest algorithm as well as both ITL and CINFO learning methods. The best results are highlighted in bold. As the results show, ITL process improves the performance of IForest by combining the scores from various subsets of the feature space. The best AUC results are achieved for *AD* dataset for which the results of ITL shows a dramatic improvement (around 22%) compared to



(a) AD

(b) Secom

(c) DoS

(d) U2R

(e) Seiz

Figure 4.3: AUC comparison for IForest when applied on input data with all features and with ITL Reduced set of the features. The results are average AUC over cross-validation folds.

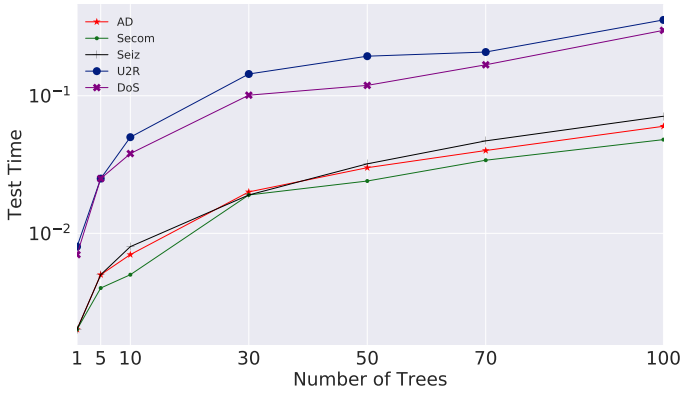
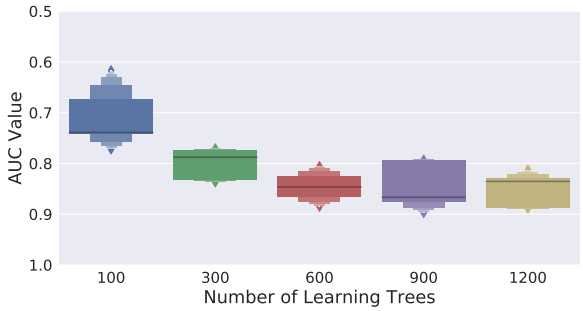


Figure 4.4: Run-Time for the Testing of cross validated results on the reduced features. Logarithmic scale is used on y axis.

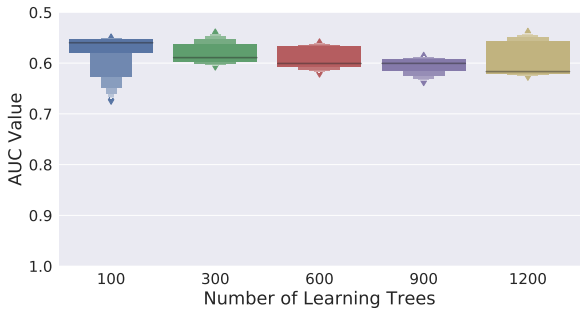
the base method. This is a result of the higher ratio of noisy features in this dataset. In a comparison to CINFO method, same or better performance is observed for 4 of the 5 datasets. The only exception is *Secom* where ITL shows improvements compared to the base, but not as much as the CINFO. This could be attributed to the greedy removal of features in vertical partitioning of ITL as we explained in Section 4.4.1. Since the results for *DOS* and *Seizure* are very high, even with the base IForest (higher than 95%), we do not expect to see too much improvement. However, ITL still shows comparable or improved AUC while achieving a reduction of about 8% and 43% in the size of the feature set. In general, ITL shows improved results as well as a reduction of the features between 9% to 97% compared to the original set. These results are especially important when the quality of reduced features is investigated for detection of unseen anomalies. Therefore, in the following, we further study the effectiveness of the reduced subset of features produced by ITL in anomaly detection results.

ITL with Reduced Features

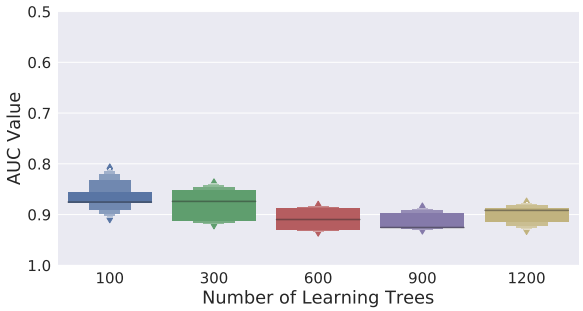
To validate the efficacy of reduced subset of features on the detection capability of IForest algorithm, a series of experiments are conducted based on the k-fold cross validation. The 5-fold validation is used to train IForest model on 4 parts of the data when all features are included in comparison to the data with the reduced features from ITL process. The AUC value of validation part is reported in Figure 4.3. The results are presented for



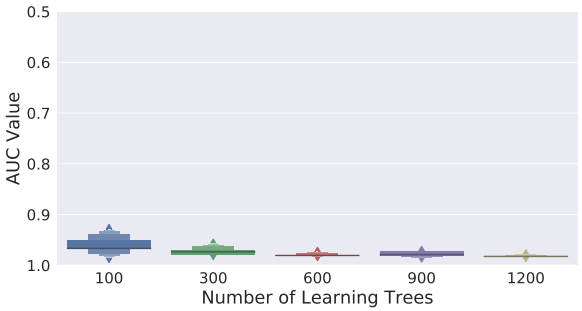
(a) AD



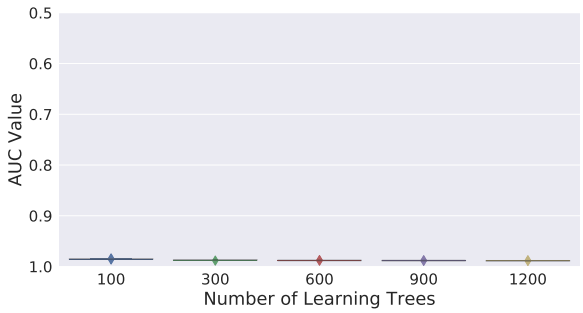
(b) Secom



(c) U2R



(d) DoS



(e) Seiz

Figure 4.5: AUC value distribution for ITL Reduced Features in Training. This plot shows the sensitivity of ITL process to different numbers of the learning trees.

different number of trees from 1 to 100. As we can see, reduced features can achieve or improve AUC value compared to the full set of the features for a range of number of trees in all datasets. The interesting observation is that the reduction in the number of trees has less impact on the performance, especially for the reduced set as shown in Figure 4.3. For example, even with 10 trees the results are very close to the performance of the algorithm with default parameters (100 trees). This improvement can be attributed to having less number of features to be explored during random selection of the features. In other words, having a subset of the features learned through ITL process, one can achieve the improved results with less number of trees. The reduction of features as well as the number of trees can help to reduce the complexity in terms of the memory and runtime requirements. Figure 4.4 shows the running time taken for a variety of tree numbers. As we can see, the reduction of number of trees can hugely impact the testing time. This is highly important for scenarios that the testing should be performed regularly. These results indicate ITL approach as a potential choice to be employed by real-time applications where new incoming stream of data requires quick online tests for identifying possible problems.

During ITL learning phase, the number of iTrees in each ensemble is a parameter which should be decided for each iteration. In order to have a better understanding of the sensitivity of ITL to this parameter, we run ITL several times for a range of values for number of trees. Figure 4.5 shows AUC distribution of each set of the experiments for all datasets. As the results show, ITL is sensitive to this parameter. However, AUC values show improvements with increased number of trees and are stable for numbers larger than 600. Practically, we found that a value between 600 to 900 trees is sufficient in most cases to have a good trade-off between accuracy and training complexities in terms of the memory and runtime.

Time Complexity and RunTime Analysis

Algorithm 2 presents the main steps of the ITL process. The main while loop (Line 2) continues until the termination condition of having zero unseen attributes is met. The loop typically converges in less than 5 iterations. Lines 3-11 build IForest models and

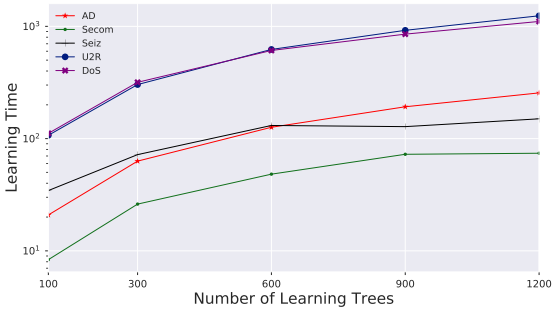


Figure 4.6: Total Run-Time of learning phase of ITL. Logarithmic scale is used on y axis.

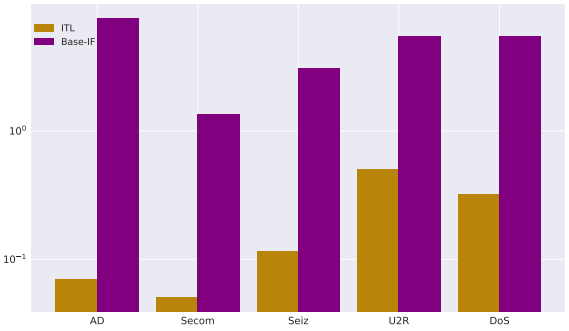


Figure 4.7: Comparison of modelling times for ITL-produced features with reduced number of iTrees (yellow) and base IForest algorithm (Purple) with default parameters. Logarithmic scale is used on y axis.

filter high-rank instances based on the predefined threshold. Considering the IForest trees as the base structure for these steps, it takes $O(t\psi log(\psi))$ for constructing where ψ is the number of selected subsamples and t is the number of constructed trees. If there are N testing points, it requires $O(Ntlog\psi)$ for determining anomalous points and $O(Ktlog\psi)$ for updating frequency profile of the features where $K << N$ (Line 5) is filtered anomalies (Worst case complexity is order $O(t\psi(\psi + N))$). Therefore, we expect a linear complexity with regard to data size.

IForest is shown to have a very fast and memory efficient runtime for both modeling and testing purpose. In order to have a clear understanding of the ITL contribution to make this process even faster, a series of execution times with respect to the number of learning trees are presented. Figure 4.6 shows the learning time in ITL, where the main feature refinements are done by constructing iTrees and creating new subset of features. The diagram shows the learning time for a variety of tree numbers. As it is mentioned before, 600 to 900 usually is enough to have a sufficient exploration of feature space for target datasets. When the learning phase of ITL is completed, the anomaly detection is done by modelling iTrees with extracted features. To have a better comparison of execution times, Figure 4.7 compares modelling time of ITL-learned features with reduced number of Trees with the base IForest without feature refinements and with recommended number of trees in the literature. As we can see, ITL process makes a dramatic decrease is modeling times by helping to decrease the number of features/trees which makes the construction of iTrees and training step much faster. It should be highlighted that this reduction is achieved by keeping or improving the detection accuracy as it is shown in Figure 4.3. However, the feature refinement process of ITL as shown in Figure 4.6 is the cost of achieving these results. But the learning phase is one-time process which is performed off-line and the final subset is used for subsequent anomaly detection task which is significantly improved in terms of both modelling and testing times as shown in Figure 4.7 and 4.4, respectively. Considering the context of one application, learning phase can be done with a low frequency and as a background process. Therefore, systems that require regular updating of their performance models can highly benefit from time/memory reductions of this process.

In conclusion, ITL shows that by targeting the main contributing features which iso-

late the instances in iTrees we can reach a refined set of the features that can be used by less number of trees to create a model with better results.

4.5.3 Strength and Limitations of ITL Approach

IForest algorithm, as described in Section 4.3, is designed to detect anomalous objects by ensemble of binary trees from input data. ITL tries to take the advantage of this mechanism to extract information about relevant features that better isolate instances. Since the core of the ITL is iTree data structures from IForest, the same advantages of random based sampling and feature selection are equally applicable to ITL. Moreover, it can be used as a pre-processing step to learn a reduced set of the features for any other anomaly detection algorithm. ITL is a promising method for real-time applications as high detection accuracy can be achieved with small memory and time complexity. Another strength is that ITL is an unsupervised method and does not require a training data containing the anomaly annotations.

Similarly, ITL inherits same drawbacks as the base algorithm in detecting local clustered anomalies [149]. This can affect the filtering of instances, when the assumption is made that there are a majority of anomaly instances at the top of the ranking list. Adaptive, data-dependent configuration for parameters such as maximum height of Trees or customized split point selection for node constructions may help to reduce this effect, but requires more pre-processing and knowledge on statistical characteristics of anomalous data.

4.6 Summary

Advances in monitoring and storage capabilities provide high volume of information on the performance of application and systems to be used for anomaly and fault analysis. These require real-time analysis of data to quickly identify problems and take appropriate corrective actions. However, high-dimensional data can adversary affect the traditional measures of anomaly detection such as distance between instances in terms of the efficacy and time complexity. More recent approaches such as isolation-based tech-

nique try to directly target the main features of anomalies as being different and rare. Therefore, in this chapter, we introduced an iterative learning framework (ITL) for the refinement of features and improvements of anomaly detection process. ITL is designed based on an idea that isolation-based generated tree structures can give insights on the importance of the features. Therefore, the learning phase of ITL is based on the knowledge from iTree structures which are binary trees constructed by random selection of the features from domain problem. The assumption is made that the features on the short branches of iTree can be used as a reference to identify relevant features to the detection of anomaly instances. The learning is based on the iterative removal of the noisy and irrelevant features in terms of their importance for isolating anomalies to generate a final subset of the features to be used for anomaly detection. The experiments show that the anomaly scores from IForest algorithm on generated subsets of the data at each iteration can be combined to create more informative set of the scores in terms of the detection capability of anomaly instances. Moreover, the experiments on five benchmark datasets demonstrate that with the reduced set of the features and choosing a proper number of trees IForest can achieve better results in terms of the detection accuracy while reducing the complexity of algorithm.

In the last two chapters, we addressed the problem of anomaly detection for both heterogeneous web-based time-series and also high dimensional data. However, they do not demonstrate how this information can be utilized by resource managers to improve the performance of the system in terms of the resource utilization and quality of the service. To have an efficient resource scaling and performance management, the manager daemons should be able to utilize anomaly detection procedure pro-actively to have enough time to decide and react upon performance degradation. As a result, in the next chapter, we propose a framework for scaling of resources paired with a performance prediction based anomaly detector that allows a combination of vertical and horizontal scaling depending on the type of the detected anomalous events.

Chapter 5

An Anomaly-based Cause Aware Auto-Scaling Framework for Clouds

An anomaly detection module helps the cloud system to identify anomalous events in the environment. However, to avoid or alleviate the performance degradations, proper scaling actions should be triggered. This chapter addresses the second and third questions of this thesis as introduced in Section 1.2 by proposing a 2-level cause aware auto-scaling framework; this framework leverages two types of resource management solutions, horizontal and vertical, as the corrective actions when the performance is degraded. We show the effectiveness of vertical scaling strategy as a quick solution for cases that a VM is exposed to the local anomalies, while horizontal scaling solutions can be used for system wide anomalies to change the number of VMs in the system.

5.1 Introduction

Elastic resource management of cloud systems offers the providers the ability to dynamically adjust the resources based on the type and number of requests from the users. For example, existing auto-scalers such as Amazon elastic auto-scaler [4] enable the system to dynamically add and remove Virtual Machine (VM) instances as a response to the observed performance degradations in the system. Workload fluctuations targeting hosted applications are one of the main underlying reasons for these performance problems. This is a highly important observation, especially for the large scale web application

This chapter is derived from:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ACAS: An Anomaly-based Cause Aware Auto-Scaling Framework for Clouds, *Journal of Parallel and Distributed Computing (JPDC)*, Volume 126, Pages: 107-120, ISSN: 0743-7315, Elsevier Press, Amsterdam, The Netherlands, April 2019.

systems where the interaction between users and web servers can change frequently, affecting the pattern of workloads and resource requirements. On the other hand, there is a variety of problems that can happen locally in one VM such as a bug in the application code, resource bottlenecks or hardware faults. This type of problems can affect the local performance of the VM adversely. Distinguishing these faults from system wide problems can help auto-scalers to make more informed decisions by focusing on the solutions that target directly the root cause of the anomalies. To achieve this goal, we can divide the data-aware resource management problem into two main subproblems, data analysis and resource management by auto-scaling, that can be dealt with separately.

First, it should be mentioned that different types of performance problems in the VMs usually leave distinctive signs in the performance indicators of the machine. Therefore, continuously monitoring the behaviour of resources by collecting the values of important attributes provides system administrators a valuable source of the data that can be analyzed to have timely information about the performance of the system. The solutions proposed in Chapter 3 and Chapter 4 offer the necessary concepts and tools to analyze these collected data and find interesting patterns of unexpected behaviours or anomalies encountered by the system.

The second part of the problem focuses on the auto-scaling solutions to be triggered when a performance problem is identified by analyzing the collected data from the system. There is a variety of resource management solutions including horizontal scaling, elastic VM management, migrations, resource contention management, etc for alleviating the performance degradations. However, when the scaling actions should be triggered and which type of the action is selected are different challenges which are investigated in this chapter.

With regard to the aforementioned challenges, this chapter focuses on the last two phases of MAPE loop (Planning and Execution) by proposing an Anomaly and Cause Aware auto-Scaling (ACAS) framework consisting of three main modules, monitoring, data analyzer, and resource auto-scaler which exploits two types of the resource adjustment policies, horizontal and vertical scaling. ACAS includes a proactive anomaly detector and a mapper between performance anomaly types and corresponding resource scaling decisions. In this work, we focus on the local anomalies such as CPU and mem-

ory bottlenecks as well as system wide load problems that can affect the performance of the applications. The proposed proactive, unsupervised anomaly detector is able to predict performance data of the VMs and identify future anomalies of the system. We have also developed a strategy for deciding when the anomaly detection models need to be updated to reduce the recurrent model training overheads. The proposed solution can achieve better scalability by breaking down the problem of performance management to two levels of local and global layer. An extensive set of the experiments are performed targeting both types of local anomalies and global load problems. The experiments show that distinguishing between VM specific anomalies and system wide load problems help auto-scaler to take advantage of fast vertical scaling policies to increase bottleneck resources for one VM while the proactive anomaly detection helps to trigger early system wide horizontal scaling actions to reduce the number of SLA violations.

The rest of this chapter is organized as follows: Section 5.2 introduces some of the existing works in the field of data-aware resource management. Section 5.3 presents the motivation and an overview of the approach. Section 5.4 presents the details of learning algorithms and explains communication among the modules. Section 5.5 presents the experiments and the results and finally, the chapter findings are summarized in Section 5.6.

5.2 Related work

The idea of utilizing data learning techniques for the performance analysis in the cloud has been of great interest to the researchers in recent years. The work presented in [136] investigates the feasibility of Isolation-Trees based anomaly analysis to detect anomalies in data from IaaS data centers, focusing on the behaviour of the algorithm to the presence of seasonality/trends in their dataset. ACAS also leverages the same concept of Isolation-Trees in the anomaly detection part of the problem. However, ACAS is a complete framework that covers the problems of online learning and model updating, root cause analysis and resource management modules. [15] proposes a method for long-term load prediction in Google data centers, considering load as the main factor involved in resource management solutions. Another work presented in [16] considers a

single attribute, number of required processors at a certain time, for resource utilization estimation. [18] presents a regression based workload prediction framework to improve the utilization of the resources while reducing the cost. To achieve this goal, they use the knowledge from workload prediction to decide the time and amount of resources to be changed in the system, considering both types of vertical and horizontal scaling. [17] combines workload prediction and reinforcement learning to find the best configuration for VM resources. The feedbacks from application performance and resource utilizations are used to calculate the reward and update the resource configuration strategy for better selection of future actions. Compared to our framework, the aforementioned works address the problem of resource management by focusing on the workloads as the only influential factor for performance analysis and ignore other sources of performance problems in the system. [86] follows a more systematic approach to the problem of VM management in the cloud by modeling the problem as a feedback-based control approach. The Proportional-Integral-Derivative (PID) based controller is designed to manage the number of VMs in the system, aiming at keeping the service quality in accordance with the agreement levels. [162] designs a reinforcement learning approach to gradually learn from the environment and decide on the VM level scaling of the system to alleviate the performance problems occurring due to the load fluctuations in the system. Different to our model, these works consider the management of resources only at VM level by changing the number of VMs in the system.

[20] presents an automatic anomaly identification technique for adaptively detecting performance anomalies such as Disk and Memory related failures. Proposed method investigates the idea that a subset of the principal components of metrics can be highly correlated to specific failures in the system. BARCA, proposed in [23], is another framework for online identification of anomalies in distributed applications which divides anomaly detection process into two steps. First, a one class classifier is employed to distinguish normal behaviour from unexpected ones. Second, a multi-class classifier is used to separate different types of anomalies from detected abnormal behaviours. [25] investigates proactive anomaly detection in data stream processing systems. The proposed solution includes a phase of predicting resource utilization and then applying an anomaly identification algorithm on the predicted values. The target anomalies are injected and

Table 5.1: Related works on cloud performance management

Work	Data Analysis Method	Resource Management	Proactive	Unsupervised[1]	Vertical Scaling
[136]	IForest (**AD**)[2]	X	X	✓	-
[15]	Bayes Model (Workload Analysis)	✓	✓	-	X
[86]	Control Theory	✓	X	-	X
[162]	Reinforcement Learning	✓	✓	-	X
[23]	SVM (**AD**)	X	✓	✓	-
[25]	Markov models, Bayes Classifier (**AD**)	X	✓	X	-
[26]	Markov models, Bayes Classifier (**AD**)	✓	✓	X	✓
[76]	Threshold-Based Rules	✓	X	-	✓
[121]	Threshold-Based Rules	✓	X	-	✓
[17]	Reinforcement Learning	✓	✓	-	✓
[27]	Self-Organizing Maps (**AD**)	X	✓	✓	-
Proposed work (ACAS)	Isolation-based Trees (**AD**)	✓	✓	✓	✓

[1] This column is applicable for the works with the focus on anomaly detection
[2] Anomaly Detection

the training is done on a labeled dataset of different anomaly occurrences in the past data. Although these works focus on the same problem of anomaly detection, how this information can be used for resource management is not investigated. Alternatively, [26] addresses the performance problem by integrating a 2-dependent Markov model as the predictor and tree-augmented Bayesian networks (TAN) for anomaly detection. Based on the knowledge from learning algorithm, they apply some type of the vertical scaling or migration to minimize the performance degradation. [76] focuses on the cost effectiveness of vertical scaling approaches and proposes a threshold based scaling strategy to combine different scaling approaches including self-healing, fine-grained resource scaling and VM level scaling to meet QoS while reducing cloud providers' costs. [121] addresses the problem of shared memory management among multiple VM with the over-subscription approach and elastic VM technique. A threshold based strategy on the value of memory related metrics is utilized to trigger the memory adjustment actions while live migration is used to avoid the SLA violations when the total memory

demands of the VMs exceed the available memory of the physical machine. In contrast, our proposed work ACAS focuses on the effectiveness of horizontal and vertical scaling policies by leveraging the capabilities of unsupervised learning approaches for situations that system is exposed to the local and load related anomalies. Another study by [27] investigates unsupervised behaviour learning problem for proactive anomaly detection. The proposed framework leverages Self-Organizing Maps (SOM) to map a high dimensional input space (performance metrics) to a lower dimensional map without losing the structural information of original instances. In contrast, we show that resource management process can make use of the knowledge from proactive anomaly detection and root cause identification to address the specific anomalies occurring in each VM.

Table 5.1 compares the above mentioned works by highlighting the main components and characteristics of the proposed solutions for the problem of performance analysis and management in large distributed systems.

5.3 Preliminary

In this section, we explain the motivation and an overview of our approach.

5.3.1 Motivation and Approach Overview

Virtualization technologies are the core concept in the functionality of cloud models. The possibility of running many VMs and applications on one physical host brings new opportunities, as well as adds more complexity to the design of these environments.

Efficient Resource management concept is highly challenging due to the inherent dynamic nature of cloud environment, where we can host different range of applications with a variety of demands and workload types. This is especially important for the large scale web applications, in which the pattern of the incoming requests from the users can change quickly creating a dynamic environment where the configurations of resources should frequently be adjusted to satisfy the demanded SLAs. Elastic resource management, as a solution for this problem, leverages the VM based scaling of the system known as horizontal scaling. Public cloud providers offer customized policies of

horizontal scaling to satisfy the resource demands of the applications based on the load in the system. Even though the VM based scaling policy is a common approach to manage performance problems in the cloud environment, it may not be a proper solution for a different category of performance problems caused by the faults in one VM. For example, consider a situation where a memory-intensive process is started in the same VM hosting a web based application which is consuming all the available memory, ignoring the demands of the web application. Therefore, the lack of the free memory can cause performance degeneration such as longer than usual response times from the web server. The conventional scaling approaches add new VMs into the system even though the problem is not caused by the load growth from a higher number of user requests. Having the same load in the system, newly added instances incur extra costs including both resource and license costs as well as higher resource wastage due to added resources which are not utilized. Considering this scenario, there are other types of the problem that can create similar effects on the utilization of the resources. For example, it is shown that the web applications are prone to many of the performance problems which involves CPU and memory resources [8].

On the other hand, existing auto-scaling solutions such as Amazon elastic auto-scaler are designed in a way that are more suitable to track the changes at the system level. They do not consider VM based problems particularly when the changes in one VM does not have an immediate impact on the average performance of the system. One solution to address this category of problems is to have a resource management solution at fine-grained levels of control. Elastic VM architecture enables on-the-fly tuning of VM resources without turning off the VM which avoids the delays of rebooting the system.

Given the above explanations, we formulate the problem as the selection of proper resource scaling policy to satisfy the quality of service (QoS) by analyzing the state of the system to distinguish resource level bottlenecks from system wide load problems. To be more explicit about the system state, we use the following definition:

System State: State or behaviour of the system at each time is an abstract representation of operational attributes and performance indicators of the system which can be recognized in normal or abnormal/anomalous condition. The main indicators of an abnormal state are the presence of unexpected patterns or values in the load and resource

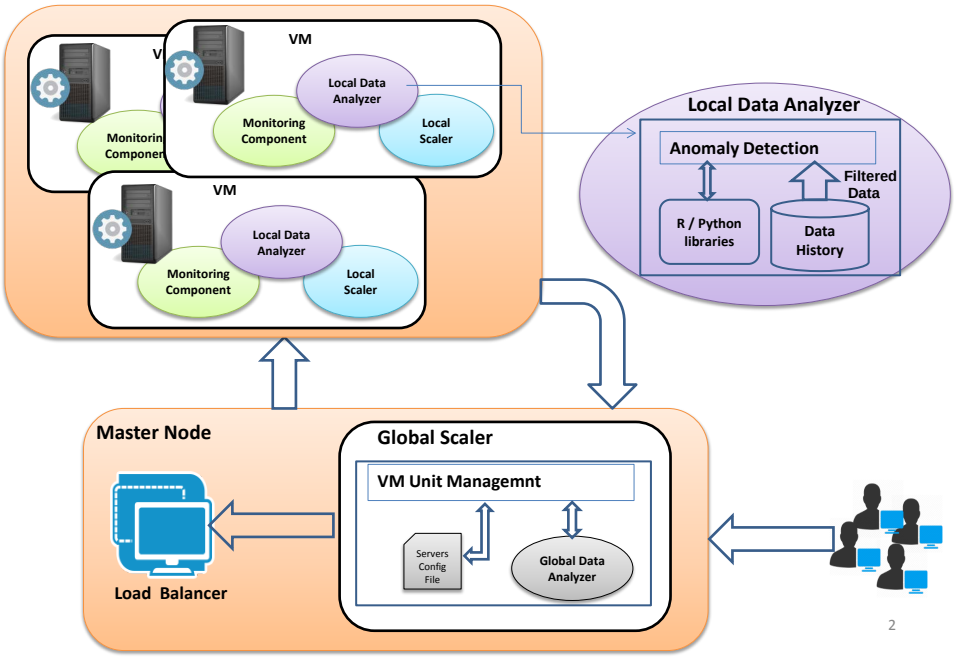


Figure 5.1: A High Level System Model

level measurements of VMs and applications.

The proposed framework addresses the resource management problem at the service provider level who has access to the VMs hosting the application to monitor system and application level metrics. In this work, we target a category of performance anomalies known as resource bottlenecks and particularly two problems insufficient CPU and memory in one VM. Therefore, by tracking resource level metrics of VMs, one can utilize vertical scaling functionality to increase the amount of RAM capacity or the number of CPU cores of one VM to quickly respond to the performance degradations of the system. When there is a system level degradation, the framework employs horizontal scaling to add new VMs into the system.

In the next section, the components of ACAS framework for cause aware auto-scaling in the cloud are explained in more details.

5.4 System Design

Figure 5.1 depicts an overview of the proposed framework and how the components work together. The framework is modeled based on a web based application with the application and database servers hosted on the cloud VMs. These applications are known for the exposure of many performance degradations caused by the changes in the workload or CPU and memory related faults. However, the definitions are generic and can be applied to any distributed application. The components of the application can be distributed on different VMs, while each VM has its own monitoring component, data analyzer and local scaler modules installed. The data analyzer box in the Figure 3.2 shows the details of the local analyzer module on each VM. The scaling decisions are performed at two levels, local and global. The local scaler is responsible for the vertical scaling decisions at one VM, while the global scaler performs horizontal scaling decision in the system. The global scaler and the load balancer are parts of a separate master node which plays as the central broker for the whole system. Therefore, the incoming requests are distributed among existing VMs (application servers) based on the load balancer configuration and registered servers at the master.

Each VM monitors the performance of its own resources and collects a variety of attributes such as CPU and memory utilization, and disk I/O rates which can model the state of the system. During regular intervals, collected data are sent to the local data analyzer to be processed for the possible signs of performance problems occurring in the near future. Therefore, at the first step, future values of each metric are predicted. There is a wide range of algorithms that can be used for the prediction and modeling of time series data. We have tested two algorithms ARIMA and feed-forward Neural Networks (NN) for this step and finally selected NN due to the observed stability of its predictions in the presence of the noise in our dataset. NN is utilized to generate a separate model for each metric and predicts the future values based on the learned models from the past observations of the system. Upon receiving the newly predicted values, the anomaly detection algorithm calculates an anomaly score for each observation and sends a new alert if the new score exceeds the threshold θ. Before proceeding, we should remark that anomaly detection module considers every deviation in the val-

Table 5.2: Description for Notations

Notation	Description
K	Minimum number of alerts before an anomaly record is created
w	prediction window size
lw	Number of observations in prediction learning window
tw	Number of observations in training window
L	Minimum number of violations before the threshold based approach (baseline) starts an action
r	Number of attributes for each observation
LI	Log Time for monitoring system to record a new observation
θ	Threshold for anomaly score. Values greater than θ will be considered as anomaly
X^m	A record of monitored metrics (attributes) from environment
th_i	Usage threshold for attribute i
f_i	An indicator of anomalousness of attribute i
S	Anomaly scores
ψ	Number of randomly selected samples from input instances as the input of IForest algorithm

ues of the attributes from the past state of the system as an anomaly which is reflected in the calculated anomaly scores. However, from the service providers perspective, a performance anomaly is important when it shows a possible breach of the SLA objectives; otherwise, it can be ignored. Therefore, to be clear about an anomaly event which is considered by the resource management module for taking the corrective actions we pursue the following definition in the next sections:

Anomaly Event: A continuous change in the behaviour of the system which is reflected as unexpected trends in the values of the monitored attributes of the VM while at least one of the metrics shows the possibility of breaching the threshold for the maximum accepted utilization.

The first part of the aforementioned definition is handled by the anomaly detection

module to detect the attributes that show a transition in their state based on the details provided in the Section 5.4.1. The second part of the definition confines the performance anomalies to the anomaly events that are breaching the performance thresholds. This part is considered by resource management module as described in the Section 5.4.2.

During anomaly detection phase and at the time of observing anomalous behaviour, the system asks the cause detection module to analyze the state of different observed metrics and find a possible cause for detected anomalies. The suggested causes of the problem from this module are used as additional knowledge in the auto-scaler components to help them make more informed decisions regarding the scaling policies.

The results from anomaly detection module are sent to the local and global auto-scaler components. The local scaler is responsible for resource configurations at VM level also known as vertical scaling policies. In contrast, the global scaler is aware of the state of the whole system and is responsible for changing the number of VMs in the system known as horizontal scaling policies. Algorithm 3 shows a summary of the main steps of ACAS framework at the local and global level. The details of these steps and the priority of different scaling policies are explained in the following algorithms and subsection. A list of all the notations used in following sections are listed in Table 5.2.

5.4.1 Anomaly Prediction based on Isolation-Trees Models

Given one VM measurements, the goal is to find if the collected values show a different pattern compared to the past behaviour (lines 3-7 Algorithm 3). Therefore, having a sequence of past observations from one VM, an ensemble of Isolation-Trees is generated using the IForest algorithm. After the training is done and each VM has the initial models of its performance, the anomaly detection process starts to analyze the new measurements collected from the VM. Algorithm 4 shows the sequence of required steps for the process of anomaly prediction in ACAS. This process is called regularly to check the recent performance of the VM.

In order to give the system enough time to trigger auto-scaling actions, we need to detect anomalies in the future data. Therefore, the first step is to predict the future values of each metric for the VM (lines 1-2). NN algorithm is exploited as the prediction

Algorithm 3: Cause Aware Resource Scaling in ACAS

input : $V = (VM_1, VM_2, , ..., VM_M)$: A list of all registered VMs in the system

while *The system is running and in the beginning of performance-check interval* **do**
  for $VM_i \in V$ **do**
    `/* This part of the code is executed locally in each VM */`
    if VM_i *has not initialised the IForest models and there are enough data collected for training* **then**
      Initialize IForest models for VM_i ;
    end
    Collect the recent monitored values for different metrics of VM_i
    Call Algorithm 4 on the collected observations to predict future data and find the possible performance anomalies and suggested causes;
    call Algorithm 5 to check if VM_i requires a new vertical scaling to be done by local scaler; If scaling is done, VM_i goes into a locked state for a predefined time.
  end
  `/* This part of the code is executed in the master node */`
  Initialize all indicators in f_i to 0;
  for $VM_i \in V$ **do**
    if VM_i *is not in a locked state and is moving to critical condition based on the Algorithm 6* **then**
      $f_i \leftarrow 1$
    end
  end
  Decide on a new horizontal scaling action based on the information provided by f_i;
end

function (f_p) to forecast the w values of each metric based on the recent measurements from the system. Predicted values are fed as the inputs to the trained models which calculate an anomaly score for each predicted record. The anomaly scores show the degree of abnormality of the observations compared to the data used in training phase (line 3).

It should be highlighted here that we expect to encounter cases where ACAS may miss some of the anomalies due to the wrong measurements or wrong predictions resulting from the dynamic nature of the target environment. Therefore, ACAS also considers more reactive mechanisms which try to adjust the scores of anomaly points when a violation in the system is detected. To make this point clear, let S_i be the score for the prediction P_i. ACAS checks if S_i actually reflects the violation observed at time t_i and if it does not (meaning that $S_i < \theta$ and $P_i \geq th_i$), it deliberately increases the score S_i to a higher value so other components of the framework handle situation as a new anomaly state.

Model Updating

One question to be answered is how the system decides to update the anomaly detector models. The inherent dynamicity in cloud workloads and the possibility of different types of failures highlight the importance of updating models so they can show the most recent state of the system. In this regard, three different states of the system are distinguished as follows:

- **Transition State**: The system is recognized as in transition if it meets two main conditions. First, newly observed values differ from the past training data in the patterns and/or values. Therefore, we expect to see higher anomaly scores calculated to show the abnormality of recent behaviour compared to the historical records. Second, the system has not reached a stable state, meaning that a continuous change of the variables is still observable. The focus of this work is on the transitions which cause the average values of the attributes to change with the assumption that the patterns remain unaffected. For example, consider a situation that an incremental trend is continuously impacting the values of one of the

Algorithm 4: Anomaly Detection

input : $D = (X_1^m, X_2^m, ..., X_{lw}^m)$, $X_i^m \in R^{1\times r}$: A matrix of lw records, each record including measurements for r features
Parameter: w: Prediction Window
θ: Anomaly Score Threshold
output : (Anomaly Alert, Cause of Anomaly)

$c \leftarrow -1$
/* Prediction function f_p is used to predict future values of data. X^m corresponds to the Measured data and X^p presents Predicted data. */
$(X_{lw+1}^p, X_{lw+2}^p, ..., X_{lw+w}^p) = f_p(X_1^m, X_2^m, ..., X_{lw}^m)$
$S_i = AnomalyScore(X_{lw+i}^p), 0 < i \leq w$: Find the anomaly scores with IForest algorithm. Then, check if these scores should be adjusted based on a reactive approach if some violation is already happening in the system.
$anomalyDetected \leftarrow (Count(S > \theta) > Length(S)/2)$
if *anomalyDetected* **then**
    Initialize all indicators in f_i to 0
    for *feature* $i \in D$ **do**
        if *system is in changing state on dimension* i **then**
            $f_i \leftarrow 1$
        end
    end
    Decide about updating the models based on the information provided by f_i.
    Identify the cause of abnormality and assign it to c.
end
return ($anomalyDetected$, c)

attributes in the system.

- **Changed State**: The system has reached the changed state when the new observations show deviations compared to the recorded data used for the training. However, the system has reached a stable condition meaning that no significant changes in the average values of the attributes are detected. In terms of the conditions mentioned for the transitions state, a system at changed state satisfies the first condition only.

- **Normal State**: The system is at the normal state when none of the above conditions is satisfied, meaning that the average values of the attributes for recent observations do not show significant changes compared to the training data. As a result, the calculated anomaly scores do not indicate any abnormal behaviour demonstrating a stable environment.

The anomaly detection module decides to update the model if it finds the corresponding VM is at the *changed* state (lines 6-12). The reason is that, at this state, the high number of anomaly alerts shows the previously trained models are not representing the current state of the system. Moreover, the system has reached a new stable environment and new models are required to enable the anomaly detection module to perform in accordance with the changes. The updating procedure continues until the new models correctly reflect the new state or another transition in the system starts. It should be mentioned that ACAS does not consider *transition* state a proper time for updating the models as some of the attributes are showing significant changes in their values and new models quickly become obsolete, resulting in many unnecessary updates.

Cause Identification

The cause detection procedure tries to provide some knowledge about the possible resource level root causes of the performance problem to help the scaling modules make a more informed decision about the proper scaling policies. Therefore, if the output scores from anomaly detection module show a possible anomaly is occurring in the VM, the next step is to identify the underlying reason for the problem. The category of changes

addressed in this work are the ones that impact the average values of the attributes with an increasing or decreasing trend. Therefore, to find an attribute with a trend in the values, we follow an approach which fits a regression line on the data and calculates the slope of the line as a measure of the existing trends in the data.

One point worth noting here is how to distinguish load problems from other local anomalies. One observation to be followed is that when the performance of the system is impacted due to the changes in the incoming workload, we expect to see more than one attribute affected and changes their state. Accordingly, ACAS checks whether most of the attributes in the system are recognized at the *transition* state simultaneously and then flags the anomaly as a load problem.

5.4.2 Resource Management Module

The management of resources in a continuously changing environment requires the integration of resource configuration policies at different layers of granularity. Depending on the type of the problem and identified root causes, some policies may work better at meeting the SLA objectives such as time or cost of the solution. In this work, two policies horizontal and vertical scaling of the resources are considered. Horizontal policies address resource configuration strategies which change the number of active VMs in the system. In contrast, vertical policies are defined at finer grains of control (Elastic VMs) and adjust the amount of allocated resources based on the new demands of the VM. Since the scaling happens online and there is no need to reboot the instance, vertical scaling is much faster and does not adds extra costs for a software license or wasted resources.

Upon receiving an anomaly alert from anomaly prediction module, the framework should create a new record to flag the beginning of a new anomaly event in the target VM. However, we need to consider the transient changes in the system that may cause false alarms. As a result, a new anomaly event is recorded at time t if the current observation is showing an anomaly alert as well as all the past observations in the window $\{t\text{-}K, t\text{-}K\text{-}1, ..., t\text{-}1\}$. In other words, the system ignores the first K alarms for one VM until there will be at least $K+1$ consecutive alerts notifying an anomalous behaviour.

A proper value for K can be selected considering the trade-off between computation overheads, the stability of the environment and the performance degradation tolerance. Small values of K may cause the system to perform unnecessary checks of the performance or decide on preventive actions for many false alarms, while large values of K increases the time it takes for the system to start a scaling action in response to the performance problems.

In the proposed framework, some conditions should be met before resource manager decides on a new scaling action for the system. The following subsections and Algorithm 5 explain these conditions.

Algorithm 5: Vertical Scaling Policy

```
   input     : counter: Number of recent alerts for the VM
   input     : anomalyDetected: True if recent anomaly score exceeds threshold
   input     : cause: The root cause detected for the current anomaly
   Parameter: K: Minimum Number of Alerts to Record an Anomaly
              θ: Anomaly Score Threshold
 1 if anomalyDetected then
 2 |   counter ← counter + 1
 3 else
   |   /* reset the counter when the system is in normal state.
   |      */
 4 |   counter ← 0
 5 end
 6 if counter > K then
 7 |   if system is not in cooling period && cause ≠ Load then
 8 |   |   If system is moving toward critical condition based on Algorithm 6, start a
   |   |    vertical scaling action.
 9 |   end
10 end
```

5.4.3 Per-VM Vertical Scaling Policies

After receiving a confirmed anomaly event for one VM, the VM starts to check if some type of the resource adjustment is required. ACAS considers scaling strategy only when a performance degradation or SLA violation is observed. In this case, we consider the breach of the resource utilization thresholds as a sign of the violation of SLA objectives.

Let th_i be the threshold for resource i. If the utilization of this resource at time t is more than th_i, system records a violation of SLA starting from time t. Therefore, no corrective action is triggered if there are enough spare resources to fulfill the requests during next time intervals. One question to be answered here is that what is the best time interval to predict the future usages of resources. Since the online resource adjustments in elastic VMs become effective almost immediately, we take one time interval away from the recent observation as the prediction interval. Therefore, the framework sends back the list of all metrics that are predicted to violate their respective thresholds at the next time interval.

Algorithm 6: Identification of System Criticality

input : $D = (X_1^m, X_2^m, ..., X_{lw}^m), X_i^m \in R^{1\times r}$: A matrix of lw records, each record including measurements for r features
input : *cause*: Root cause detected for current anomaly
Parameter: LI: Log Interval

```
delay ← 0
if cause ≠ Load then
    delay ← VerticalScalingDelay
else
    delay ← HorizontalScalingDelay
end
windowLength ← delay/LI
P = (X^p_{lw+windowLength}) = f_p(X^m_1, X^m_2, ..., X^m_lw)
for feature i ∈ P do
    if P_i exceeds th_i then
        f_i ← 1
    end
end
Decide about the criticality of system based on the information provided by f_i
```

Since vertical scaling is a response to local anomalies happening in a VM, the load problem is ignored at this step and local resource adjustment is triggered if the detected problem is related to one of the resource level metrics of the VM. Depending on the metric detected as the root cause of the problem, the system decides about changing the number of CPU cores or the amount of memory capacity of the VM to prevent performance degradations in the application. After starting an auto-scaling process, the VM will enter in a locked state which means that during this time no other scaling action is

performed. The reason is that it takes some time for the system to adapt to the changes of the resources, so the first few anomaly alerts are ignored to give the system enough time to reach a stable state.

5.4.4 Horizontal Scaling Policies

A horizontal scaling policy is performed if there are no VM in the locked state, meaning that there has not been any vertical scaling in the recent intervals that can affect the state of the system. First, the state of all VMs is checked and the number of VMs which are moving toward critical condition is recorded. One VM is recognized in critical condition if at least one of the main attributes is predicted to breach the threshold in the near future. Similar to vertical scaling procedure, we consider a rough estimate of the time it takes to boot a new VM in the system as prediction interval. In other words, ACAS asks for enough time to add a new VM before the system enters the anomaly state. If all the active VMs are found moving toward the violation state, an alert to add a new VM is issued. Afterwards, the system starts a cooling period when no scaling will take place. This waiting time is required so the load balancer can detect new VM and start sending new requests to that.

5.5 Performance Evaluation

The proposed framework incorporates multiple components from resource monitoring, resource configuration and data analysis. The framework is general and should be applicable to different types of applications and workloads. However, in order to demonstrate the effectiveness of ACAS, we select web applications which are shown to be prone to many performance problems involving CPU and memory resources [8]. The main focus of this work is the performance of the application layer which can be easily affected by the behaviour of users, buggy codes or other malfunctioning applications.

To validate the framework, we use CloudSim discrete event simulator [163] which is a framework for modeling and simulation of cloud computing infrastructures. CloudSim has been used extensively for validation of cloud services and applications that can be

Table 5.3: Experiment Configurations

Variable	Description	Value
K	Minimum number of alerts before an anomaly record is created	6
lw	Number of observations in learning window	60
tw	Number of observations in training window	300
L	Minimum number of violations before baseline approach starts an action	2
LI	Monitoring Interval (Log Time)	60
θ	Anomaly Score Threshold	0.55

hard to be validated in real implementation as we need a controlled environment where one could perform analysis of the system with and without data analysis or auto-scaling methods including elastic VMs. An extension of the CloudSim is leveraged that implements an analytical performance model of 3-tier applications in the cloud and multi-cloud environments [164]. CloudSim offers both flexibility as well as the extensively validated models of reference workloads that helped us to create a near real environment.

5.5.1 Experimental settings

The experimental environment is simulated as one cloud data center hosting the application and database servers. The application servers are modeled with the initial configuration of one virtual core, 3.75 GB of RAM and Linux operating system. The VM start-up times are modeled based on the performance study done by [119].

The following experiments are based on an extension of CloudSim which models the workloads on Rice University Bidding System (RUBiS) benchmarking environment [165]. RUBIS is a benchmark that implements the core functionality of an auction site including browsing, bidding and selling modeled based on eBay.com. RUBIS follows a 3-tier web based framework consisting of the client, application and database servers.

Sessions are the unit of works defined in the RUBIS and represent a sequence of requests from one customer interacting with the application. The resource usage of each session is monitored and modeled in the CloudSim based on the work done by [164]. In total, there are 4 attributes CPU, memory, I/O usages as well as the number of sessions which are collected during each experiment for data analysis part. For the details of how the workload is modeled and validation of the extracted models you can refer to the work [164]. To implement the prediction step, we utilize *forecast* package implemented in R which models a feed-forward neural network with lagged inputs for forecasting univariate time series. The final prediction is an average of the results from 20 trained networks; each network is trained on lag-1 of all input values. Therefore, each network has one input (with a bias node), one hidden layer with one node (with a bias node) and one final output node which is analogous to AR model but with a non-linear function. The averaging on the all networks helps the prediction result to be more robust in the presence of noise.

In order to demonstrate the functionality of ACAS in resolving local performance problems with the help of fine grained resource scaling, we have also extended CloudSim framework to enable the on-the-fly changes of the resource configurations without turning the VM off. Two main resource types CPU and RAM are considered in this implementation. However, the codes are general and can easily be extended for other types of the resources. The amount of changes in each scaling action can be configured to be a percentage of the original capacity of the resource. For the following experiments, the capacity of CPU resources increases by one core (100 percent of initial configuration) while the RAM storage is increased by 20% for each scaling action. Moreover, the anomaly detection models in ACAS are generated using IsolationForest package implemented in R environment. In order to connect the anomaly detection module to the simulation environment which is developed in JAVA, we utilize *Rengine* interface which supports calling R implemented functions from Java environment.

Each experiment has a duration of 18 hours with sessions arrival time modeled as a Poisson distribution with a frequency that is defined as a function of time [164].

https://www.rdocumentation.org/packages/forecast/versions/8.1/topics/nnetar
https://sourceforge.net/projects/iforest/

In order to evaluate different aspects of the proposed framework, four cases of experiments have been run. In two cases, the behaviour of ACAS is tested in the presence of local anomalies in the VMs. In two other cases, the system is exposed to workload increases and the functionality of the framework is analyzed. Two types of the resource level bottlenecks, insufficient memory and CPU, are simulated. In both cases, we focus on the impact of increasing trends on the corresponding attribute. In order to simulate memory problems in CloudSim, a predefined percentage of the memory storage is removed from the available memory at consecutive interval times which creates an incremental trend in the used memory of the VM. For insufficient CPU, a predefined percentage of the available CPU capacity is flagged as reserved assuming a different CPU-intensive application starts running as a background process along with the target application.

The idea of performance anomaly detection has been widely investigated in the research area. However, most of them follow supervised approaches or are designed for specific scenarios or focus on data analysis part of the problem without providing the details of an integrated framework for the purpose of resource management. On the other hand, many of popular public cloud providers such as Amazon [4] use a threshold based auto-scaling approach for dynamic scaling of their resources. In the threshold based approaches, system continuously tracks the state of the resources in the system and an anomaly alert is triggered if the utilization of monitored metrics exceeds a predefined threshold. For example, a new machine is added to the system if the CPU utilization is more than 80 percent for five continuous sampling intervals. Therefore, for the comparison purpose, we have implemented the same threshold method as our baseline approach. To have a comparable experiment, the thresholds for the baseline auto-scaler is the same as the triggering thresholds of ACAS framework. In the all experiments, this value is equal to 70 percent and is similar for both CPU and memory. A cooling period of 15 minutes is considered for the baseline simulation. Therefore, no two auto-scalings are performed in a time interval less than the cooling period. Table 5.3 shows the values of parameters used in the experiments.

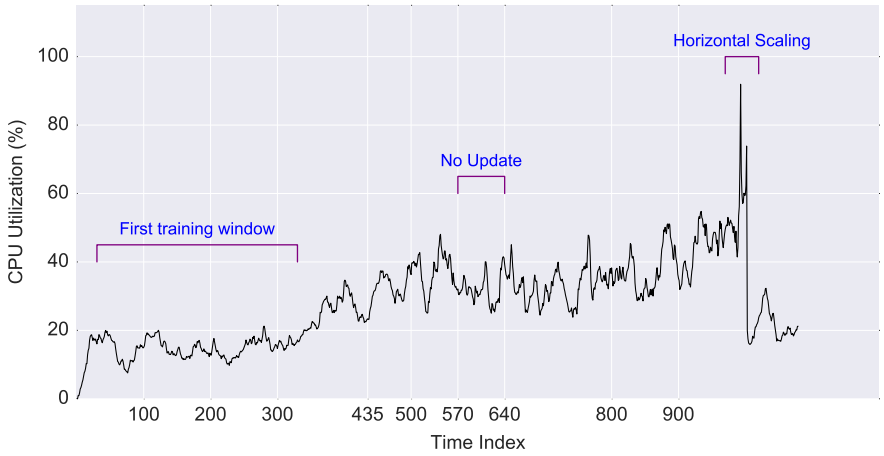


Figure 5.2: The process of ACAS on a sample workload including the first training window and one horizontal scaling action. One part of the data that is analyzed with the same models (no model update occurred during this time) is also annotated.

5.5.2 Experiments and Results

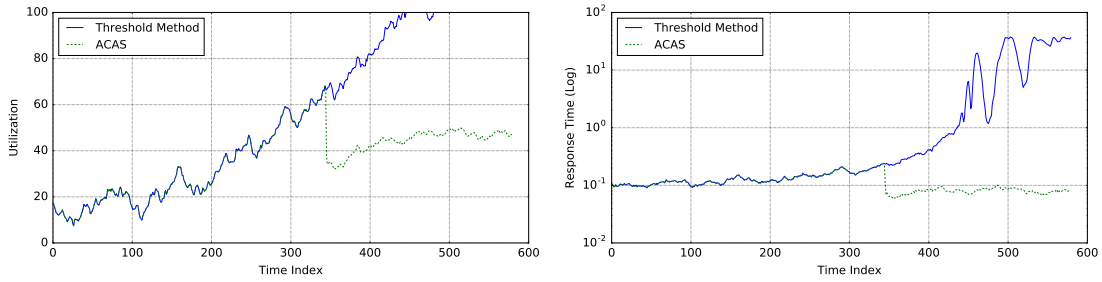
In the first experiment, we investigate the behaviour of ACAS based on a sample workload similar to Figure 5.2. The experiment starts by sending requests to a load balancer which distributes the load among application servers on a round robin basis. In order to start training the models, we follow the observations from [107] which suggests that 2^8 generally is enough to consider as the sample size (ψ) for training phase. Considering this and based on the nature of the dataset and empirical experiments to apply IForest as an online anomaly detection algorithm, 300 is selected as the training window size (tw) to be considered for the sampling and training purpose. Therefore, the anomaly module waits for the first 330 observations to pass and then initializes the first anomaly detection models by training IForest algorithm with the last 300 records as it is shown on Figure 5.2. The first 30 records are ignored for the system to stabilize. After the first initialization, anomaly detection module starts to regularly check the performance of the system by applying the generated anomaly detection models on the recent collected observations at the configured time intervals (presented as a while loop in Algorithm 3). However, depending on the state of the system and based on the definitions discussed in the Section 5.4.1, models may need to be updated occasionally to represent the new state of the system. As we can see in Figure 5.2, after the first model initialization, a

Table 5.4: Number of times that resource utilization exceeds the threshold before the first auto-scaling action is triggered. *NA* means no scaling is performed.

Anomaly Type \ Algorithm	**ACAS**	**Threshold Method**
CPU	1	NA(>100)
Memory	10	NA(>100)
System Load	5	8

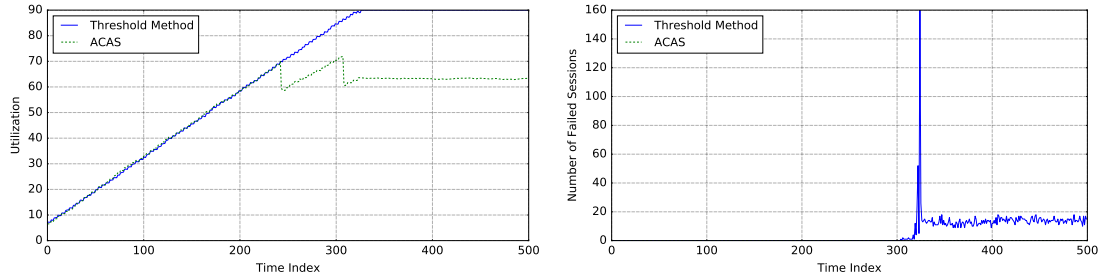
low rate increase of the incoming load is started which corresponds to a transition state based on our definitions. Therefore, the first update of the models recorded for this experiment is occurring around 435th observation when the system is identified at the end of the transition and entering a new normal state. Similarly, other updates occur occasionally during the experiment due to the fluctuations in the utilization data. However, there are also several gaps that no update has occurred during that times. These gaps are consistent with our observations of the stability of average utilization data and the functionality of ACAS which has not detected any *transition* that requires new model trainings. For example, there is no update between observations 570 to 640 or there are only 7 updates between observations 645 to 760. The reduction in the number of updates helps the system to decrease the overhead of recurrent trainings to create new models. The same procedure with similar reasoning is applicable for the next load increase, starting around observation 800, that changes the state of the system from *normal* to *transition* and also triggers an auto-scaling action which adds a new VM to the system.

The next experiments are designed to test the presence of the local anomalies in VMs. The initial configuration is done by adding 3 application and 2 database servers in the system. Then, one VM is randomly selected as an anomalous VM. For both experiments of CPU and memory anomaly, we wait for a minimum of 5 hours and then, at a random time, the injection of the anomaly in the VM is started. Table 5.4 shows the number of recorded observations that the attribute corresponding to the detected root cause exceeds the threshold. *NA* in the table means that there was no auto-scaling action in the response to the injected fault in the target VM which equals to a 100 percent violation of



(a) CPU Utilization **(b)** Response Time (Log scale)

Figure 5.3: Vertical auto-scaling for CPU bottleneck. ACAS avoids high response times by timely reaction to the predicted performance problem.



(a) Memory Utilization **(b)** Failed Sessions

Figure 5.4: Vertical auto-scaling for Memory bottleneck. ACAS avoids failed sessions by timely reaction to predicted performance problem (ACAS line for the failed sessions is zero for duration of the experiment).

the SLA. The exact number of the violations depends on the duration of the corresponding anomaly in the system which may last for hours.

For more clarification of the way these policies are reacting in the presence of local performance problems, Figure 5.3 and Figure 5.4 present the utilization of the corresponding attribute for each fault and for both ACAS and threshold methods. Regarding CPU, the ACAS has increased the number of cores by one as soon as it predicts the criticality of the CPU utilization measurements. One violation is observed in this case which is a result of the fast changes in the attribute values which the prediction function has not caught. In contrast, the threshold approach is monitoring the average state of the whole system, missing the local faults occurring at the anomalous VM. It's worth mentioning that even having a per-VM monitoring mechanism for the threshold approach can only help to trigger a horizontal scaling with the condition that the monitored values show a minimum of L violations before auto-scaler starts triggering an action. The L value should be chosen reasonably to avoid unnecessary scalings in the presence of temporal changes in the system. In our simulations, L has a small value equal to 2. However, depending on the application instability, this value can be higher which leads to even more violations. This situation is a result of the lack of the knowledge about preceding trends to the anomaly state. ACAS solves this problem by keeping the track of the patterns in the data and performing the scaling when the conditions of being in a continuous anomaly state and the violation of the threshold values are met.

The above reasoning is also applicable for memory bottlenecks. One point to mention is that the simulated RUBIS application shows a CPU intensive behaviour. Therefore, memory usage has fewer fluctuations and shows more clear change points which can be detected with higher accuracy. Figure 5.4 shows two sequential vertical scalings of memory which adds 20 percent of the initial capacity each time. The first scaling happens before any violation is observed which shows the prediction part of ACAS helps the scaler to perform a proactive action to predict the future anomaly events and start a corrective action. The results show that the memory usage drops down by 20 percent. However, the utilization continues to increase which causes the start of the second scaling action. This time, however, a few numbers of violations of the memory usage are observed. The reason is that for a small duration of time after the first scaling, the sys-

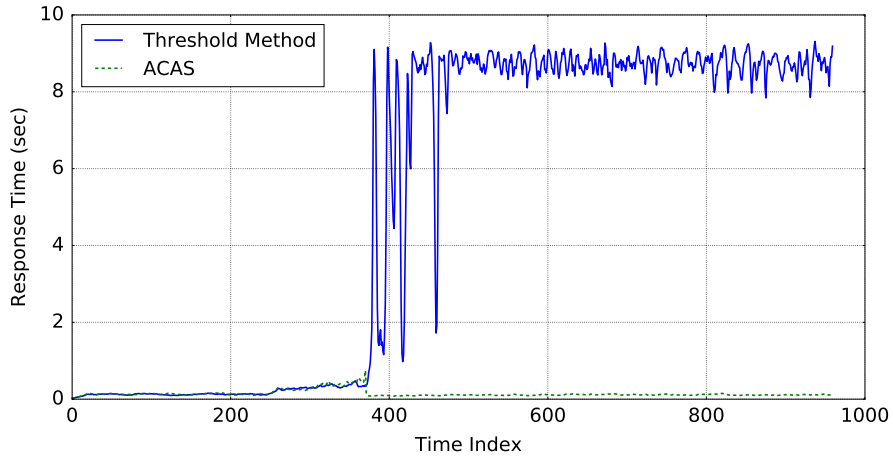


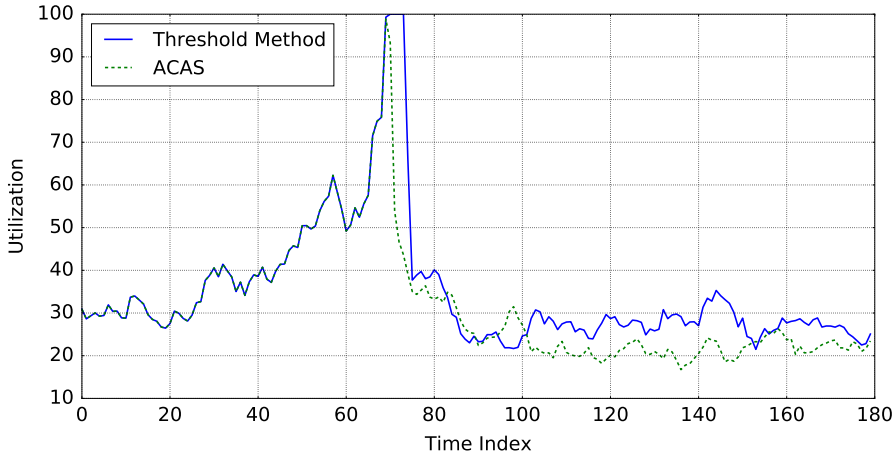
Figure 5.5: Response time of one application server when the machine is overloaded

tem is recognized in a new *changed* state which is followed by an update of the models. Therefore, the initial increases in the memory do not trigger anomaly alerts which cause the system to start the second action after some delays. In this case, the reactive part of the approach helps the system to detect the anomaly state when the violations are observed.

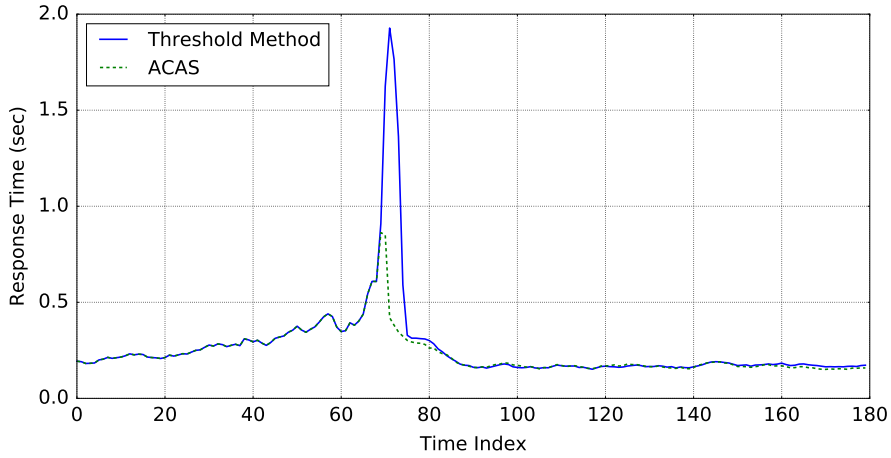
Figure 5.4 also shows the number of failed sessions for both policies. A session is flagged as failed if the VM does not have enough memory to process its requests. As the figure shows, in the experiments with the threshold method, the number of failed sessions has increased as a result of ignoring the local fault in the VM. In contrast, ACAS has properly adjusted the configurations corresponding to the bottleneck resource which avoids the unusual increases in the failed sessions.

As demonstrated by aforementioned experiments, the proactive vertical scaling helps to quickly target the bottleneck resource and reduce the number of violations by adjusting the amount of resources accordingly. This process also helps to reduce the cost as well as the energy consumptions compared to the conventional way of adding new VM machines in the system. It is also worth noticing that the local execution of anomaly detection reduces the complexity of training the anomaly detection models. As it is explained in Section 5.3 the time and space complexity of IForest algorithm is constant when the same number of training observations is used for model generation.

The next set of the experiments analyzes the behaviour of the system when the in-



(a) CPU Utilization



(b) Response Time

Figure 5.6: CPU Utilization and Response Time of one application server when the system is overloaded. ACAS is able to proactively trigger a horizontal scaling action compared to reactive response of the threshold method which causes more SLA violations.

put workload of the machines suddenly increases. Two types of the problem have been considered. The first experiment simulates an environment where one VM is exposed to an increasing workload while other VMs in the system stay in their normal state. Therefore, one VM is randomly selected and the number of the requests sent to this VM is increased. This scenario can happen in different cases such as a result of a misconfigured balancer service which assigns a higher weight to one VM. Figure 5.5 shows the impact of the load increase on the response time of the target VM for both policies. As we expect, the threshold approach is not successful at detecting the local performance problem and many violations of the response time are observed. In contrast, the local anomaly detection approach utilized by ACAS helps to identify the problem as soon as the metrics show an increasing trend followed by exceeding the thresholds.

The second experiment for the load problem simulates an overloaded system where the number of incoming requests to the balancer is increased, resulting in the increase in the resource usage of every machine at the same time. This scenario is a common case in the web applications known as flash crowds when sudden surges in the traffic to a web site causes high delays in the response time making it virtually unreachable for the users. As Figure 5.6 shows, both policies make similar decisions and add a new VM after the problem is recognized. However, ACAS approach is able to react to the problem immediately at the same time that the first breach of the threshold is detected which causes the system to return back to the normal state after 5 observations of the violation of CPU and memory metrics. In contrast, the threshold approach does not have a knowledge of the past behaviour of the system and therefore delays the triggering of the auto-scaling action for L observations. In our experiments, this value is set equal to 2 which results in about 8 violations before the system goes back to the normal state. Larger values for L, more SLA violations in the system.

Finally, a set of the plots presenting the relation between anomaly scores and model updates are shown for a sample experiment in ACAS framework. Figure 5.7 shows the CPU utilization of one application server. The marked points are the observations recognized as anomalies, meaning that the corresponding anomaly scores are higher than 0.55. Figure 5.8 presents a combined view of the anomaly detection process for the same workload, including detected anomaly points along with the anomaly update times.

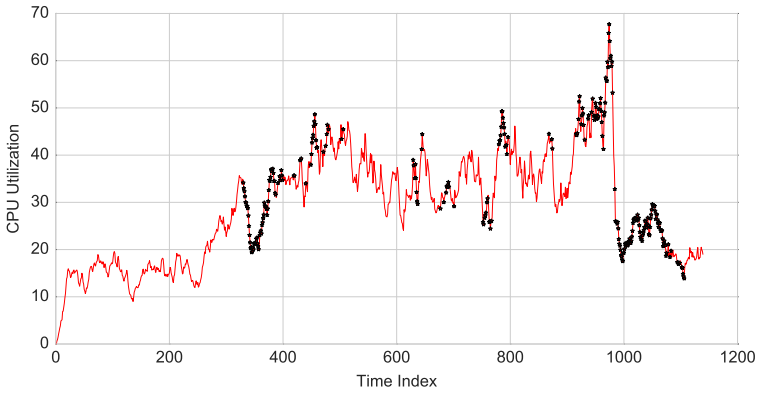


Figure 5.7: CPU utilization of one application server when the machine is overloaded. The marked points are the records detected as anomaly.

Each point at the top line shows that the observation at the corresponding time was detected as an anomaly, while the gaps between these points reflect normal or transition states of the system. The first 330 points are ignored as they are used during the training phase and detection process was not activated at that time. Similarly, each point at the bottom line shows that a model update happened at the time of the corresponding observation. As we can see, at the times that the system is recognized in the normal state, no update is occurring, meaning that the models are reflecting the current state of the system. Another observation from these figures indicates that the updates are delayed when an anomaly event is started while the system is recognized as being in the transition state. An example of this condition can be seen between observation 900 to 1100 which is reflected by the gaps among the points at the bottom line.

5.6 Summary

Elastic VMs with the accompanying knowledge from performance data analysis can bring new opportunities to offer better resource management solutions in the distributed environment. In this work, we show how fine grained resource configurations can help to improve the auto-scaling efficiency for a category of local anomalies occurring in one VM. The proposed ACAS framework utilizes a low overhead anomaly detection solution based on the Isolation-Trees and combines it with a cause identification procedure

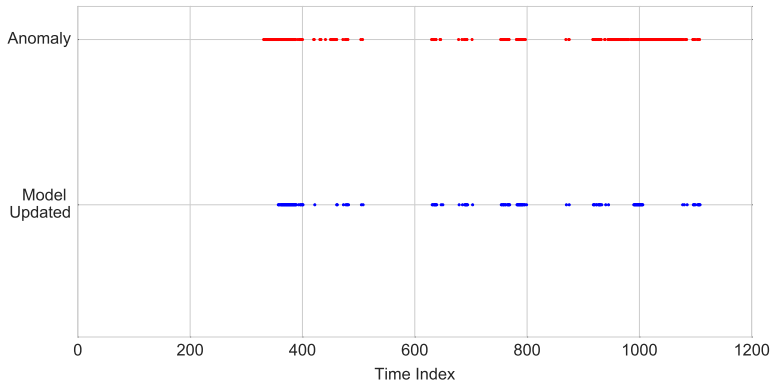


Figure 5.8: Detected anomaly points and the model update times for the duration of the experiment. Red points show the observations that detected as an anomaly. Blue points show the times that a model update occurred in the system.

to enable appropriate auto-scaling techniques taking into consideration the nature of the anomaly. The experiments show that local vertical scaling actions can efficiently respond to local anomalies in terms of the resource consumption and QoS. In contrast, performance degradations caused by load increase on all VMs can be alleviated by adding new VMs to the system.

ACAS demonstrates the effectiveness of combining the knowledge of performance data analysis with resource scaling decision makers. However, decision maker is designed as a rule-based system with if-then-else conditions and actions. As we stated in Section 1.2, the adaptability of final resource decision maker is critical to be able to manage a variety of system states with a minimum knowledge from the dynamics of environment. Therefore, the last chapter of this thesis focuses on improving the adaptability of the system with the help of gradual learning frameworks.

Chapter 6

ADRL: A Hybrid Anomaly-aware Deep Reinforcement Learning-based Resource Scaling in Clouds

This chapter addresses the second and third research questions of this thesis as explained in Section 1.2 and proposes a hybrid Anomaly-aware Deep Reinforcement Learning-based Resource Scaling (ADRL) for dynamic scaling of resources in cloud. ADRL takes the advantage of anomaly detection techniques to increase the stability of RL decision maker by triggering actions in response to the identified anomalous states in the system. Tow levels of global and local decision makers are introduced to handle the required scaling actions. An extensive set of experiments for different types of anomaly problems shows that ADRL can significantly improve the quality of service with less number of actions and increased stability of the system.

6.1 Introduction

The efficacy of resource management solutions can be interpreted from the level of user happiness; however, a combination of heterogeneity of applications, resource sharing conflicts, workload patterns and etc. can contribute to the violation of service level agreements (SLA) and users' Quality of Service (QoS). Therefore, proper scaling of resources depends on the comprehensive understanding of environmental changes and dynamic factors which can affect the performance of the system.

This chapter is derived from:

- **Sara Kardani Moghaddam**, Rajkumar Buyya, Ramamohanarao Kotagiri, ADRL: A Hybrid Anomaly-aware Deep Reinforcement Learning-based Resource Scaling in Clouds, *IEEE Transactions on Parallel and Distributed Systems (TPDS)* (under revision)

On the other hand, the workloads in cloud are dynamic and uncertain. Therefore, the prediction of future load is not easy and depends on many factors, some out of the knowledge of system administrators. Dynamic threshold-based solutions, time-series based analysis or machine learning based techniques are proposed to address these problems [18, 26, 80, 105]. However, considering the uncertainty of environment, it is critical to have a solution with a policy for updating the base assumptions, parameters and learning models. Therefore, having an updatable decision maker is an essential part to have an adaptable system with regard to the scaling of resources to ensure QoS satisfaction in presence of various performance related problems.

We have investigated adaptive learning frameworks such as reinforcement learning (RL) and how they can fit into our problem. In RL, continuous interaction of agents with surroundings develops an up-to-date knowledge base by collecting dynamic measurable metrics of the system. The knowledge is formulated as a set of the states that define an abstract representation of the target system. RL is modeled as a control loop and the gradual learning happens in a process of trial and error. This feature is especially important in an uncertain environment, where the prior knowledge is not very clear. Therefore, at each step, the available knowledge is used to select actions that may change the environment. Then, the knowledge base is updated with recent feedbacks from the environment.

While the RL paradigm seems to fit our problem, when the action should be triggered and the type of the selected actions are two main challenges that make the problem difficult in terms of the complexity and dimensionality of the state/action space. First of all, the level of resource control granularity considered in the RL can target different types of performance problems. Despite many RL based attempts in the literature, the possibility of having a range of scaling actions including vertical and horizontal for different states of the system are not investigated. Second, the majority of RL based solutions do not consider the possibility of reaching a stable state where no action is required to move toward new states. In fact, the inherent characteristic of RL which learns from the results of triggered actions in the environment along with the highly dynamic nature of cloud and constraints on available resources can push the system to constantly change its state to observe the consequences of combinations of states and actions. While recent

developments in Deep learning based RL frameworks (DRL) try to utilize the learning capability of deep networks for modeling the value of state/action pairs, their focus is more on improving the efficiency of RL in searching larger state/action tables rather than the evaluation of necessity of taking actions. Particularly, in the context of cloud computing resource management, the actions are meant to be triggered as a response to the performance problems in terms of the resource utilization and QoS. This requirement highlights the need for more customized solutions that integrate the performance related knowledge in RL decision making process.

To address the above mentioned challenges, we propose a deep reinforcement learning resource scaling framework that combines two levels of vertical and horizontal scaling to respond to the identified problems in the cloud. The proposed solution focuses on improving the adaptability of MAPE loop as discussed in Section 1.2 by designing an RL-based connection between planning decision maker and environment; ADRL utilizes an anomaly event based controller to detect the persistent performance problems in the system as a trigger for the decision making module of RL to perform a scaling action for correcting the problem. The deep learning part helps to increase the quality of decision making in large state space of the problem while the anomaly detection module addresses the timely trigger of scaling decisions. Two levels of scaling are proposed to address various types of performance problems including local VM-level resource shortage and system level load problems. Experiments of the proposed system under various loads demonstrate ADRL ability to improve the performance compared to the benchmark and state of the art approaches.

The rest of this chapter is organized as follows: Section 6.2 overviews some of the related work in the literature. Section 6.3 discusses the motivation and assumptions in our modelings. Section 6.4 overviews the basics of Reinforcement Learning architecture. Section 6.5 presents a general discussion of the main components followed by the details of ADRL framework in Section 6.6. Section 6.7 presents the experiments and validation results. Finally, Section 6.8 summarizes the results and findings.

Table 6.1: Related works on RL based cloud performance management

Work	Base RL	Resource Management Problem	Dimensionality Solution	Decision Level	Anomaly aware	Scaling Method
[166]	RL(Q-learn)	Scaling	Fuzzy states	Global	X	H
[111]	RL(SARSA, Q-learn)	Scaling	Fuzzy states	Global	X	H
[162]	RL(SARSA)	Scaling	Parallel agents, Function approximation	Global	X	H
[112]	RL(least-square policy iteration (LSPI))	Migration Management	Sparse Projection	Global	X	-
[110]	RL(Q-learn)	Migration management	-	Local (Host-level)	X	-
[167]	RL(Model-based approach)	Scaling	Decision-Tree based Models (Adaptive state partitioning)	Global	X	H
[64]	RL(SARSA)	Scaling	Model-based Environment	Local (Host-level)	X	V
[17]	RL(Q-learn)	Scaling	Parallel agents	Local (Host-level)	X	V
[168]	RL(Q-learn)	Task Scheduling	Deep RL , Multi-level Decision Maker	Global	X	-
[68]	RL(Q-learn)	VM to Server Mapping, Power Management	Deep RL, Representation Learning	Global, Local	X	-
Proposed work	RL(Q-learn)	Scaling	Deep RL, Multi-level Decision Maker	Global, Local	✓	H, V

6.2 Related Work

Resource scaling decisions are usually a response to the performance degradations of the system. However, a variety of factors at different levels of granularity from workload and application level characteristics to software and hardware functionality can affect the performance. Therefore, a proper solution should exploit updatable models and adaptable architectures to create a self-directed learning environment.

The problem of autonomous scaling of resources can be easily mapped to MAPE-K architecture (Monitor, Analyze, Plan and Execute over a shared, regularly updated Knowledge base) of the autonomic systems. Following this architecture, [80] proposes

a cost aware auto-scaling framework with the focus on possible improvements at the execution level. The planning is done based on the threshold based rules on monitored metrics to change the number of VMs in the system. While threshold based decision maker is simple and convenient in terms of interpretation and implementation, the lack of the flexibility to adapt to the changes in the environment makes that a sub-optimal solution for these types of the problems. To achieve higher adaptability at decision making level, Reinforcement Learning (RL) introduces a self-adaptable framework that can easily be matched by the phases of MAPE architecture. RL has been used for various types of resource management in cloud. [166] utilizes Q-learning as part of the planning phase of the MAPE loop. The decisions are made as a combination of Markov decision table and Q-table to decide on adding/removing of VMs in the system. A fuzzified version of Q-learning and SARSA learning is introduced in [111]. They use the fuzzy rules on the monitored metrics as a solution to reduce the number of states and as a result the size of the Q-table. In these works, threshold and rule based techniques are used to decrease the number of states, while actions are limited to the adding or removing of VMs in the system.

Megh [112] is another RL-based system which targets the energy and performance efficiency of resource during live migrations of VMs in the system. The actions are defined as selecting the destination host of the migrated VMs. They use a projection method to reduce the state space complexity of their problem to a polynomial dimensional space with sparse basis. Alternatively, Q-learning is used in [110] to schedule the live migrations of VMs. A combination of *waiting* and *migrating* actions are used to decide on the order of VM movements in th presence of network congestions to ensure having enough available bandwidth for on-time migrations. In contrast to these works, our work focus on resource scaling actions that change the configuration of resources as a response to the performance problems in the system. [167] introduces an adaptive state space partitioning technique to overcome the high dimensional state problem. The environment is represented as a global state at the beginning. Then, as more data is available, new states are created which maps the new observed behaviors of the system. This technique is especially important when the amount of training information is limited and the cost of collecting new data is high in terms of the time and operational costs. Alternatively, our

work addresses this problem by having a distributed approach and utilizing Deep Reinforcement Learning to handle local state of the VMs. VCONF [64] and VScaler [17] are two other frameworks that use the RL paradigm for vertical scaling of resources. VScaler uses parallel learning technique where agents can share their experience from the environment to speed-up the convergence. VCONF exploits neural networks (NN) learning to model the relation of (s, a) pairs with their corresponding rewards. The same parallelization technique as VScaler is also used by RLPAS for managing the number of VMs in the system[162]. The general idea of model based RL as discussed in VCONF and the concept of Deep-RL (DRL) enables the system to adaptively learn in complex problems with high dimensional space and low actions. Introduction of DRL techniques and their success in playing Atari offers new directions for the problem of dynamic, continuous time state space of resource management. Accordingly, DRL-cloud [168] is proposed to minimize the long-term energy cost of the data centers. The problem is formulated as a two-level task scheduling. The first level assigns tasks to a cluster of servers and the second phase chooses the exact VM on the selected server. Another work by [68] leverages DRL in a two-level VM allocation and power management framework. The DRL agent is used at global layer for allocating VMs to hosts while RL and workload analysis are used in local VMs to manage the power. Our model is inspired by such models, but focuses on the hybrid scalings as part of the action set as well as anomaly-based triggering of decision maker for decreasing the amount of oscillation resulted from sequential actions. Table 6.1 compares some of the RL based works in the literature considering their resource management actions and techniques for handling high dimensional state space.

6.3 Motivation and Assumptions

The elasticity feature of cloud environment which allows scaling resources dynamically based on the performance of the system brings the flexibility to handle dynamic applications with constantly changing requirements. However, the dynamic adjustment of resources in accordance with the state of the system requires self-adaptable techniques that can interact with environment and learn the effect of resource changes in a variety

of load and resource configurations. To achieve this goal, the proposed solution should be able to answer three main questions.

First question is when the decision should be made. Time-based monitoring and decision making [17, 166] is a common approach which helps the system to continuously adjust the amount of resources according to the load and performance state of the system. However, in the context of the cloud resource management, the dynamicity of environment can push the system to make unnecessary actions in response to the temporal performance problems. For example, a short spike in load can over-utilize the resources for a short amount of the time. A proper response to over-utilization is to increase the amount of resources. While this is a correct action at the time of the observation, the problem is a temporal spike and the system quickly goes back to the normal load while the amount of resources is increased which may cause under-utilized state. Second question is which types of the scaling should be actioned. Two main types of scaling in the context of VM resources are vertical and horizontal scaling. While horizontal scaling can help for general load problems, as we demonstrated in Chapter 5, local problems can benefit more from vertical scalings in terms of the performance maintenance and resource utilization. Finally, we should decide how to select an action for each state of the system. A common solution for this problem is a combination of if-then-else rules and threshold based methods that describe the system in two main states of over-utilized and under-utilized and adjust the amount of resources accordingly. However, for highly dynamic cloud environment, there are many internal and external factors such as CPU hog and memory leak problems that can affect the performance of the VMs. These types of the problems require complex rules and analyzing methods to be properly managed. Considering the constant changes of application requirements as well as the limitation of physical resources which affect the amount of available resources, self-adaptable solutions show a potential for automating the process of resource management. These systems can learn from the environment and tune their parameters, update their models and adjust their decisions based on the most recent feedbacks from the system.

Given the above explanation, we define our problem as dynamic reconfiguration of resources in cloud in response to the performance problems in the system. This work also addresses the experiences of end users and considers QoS as a measure for validat-

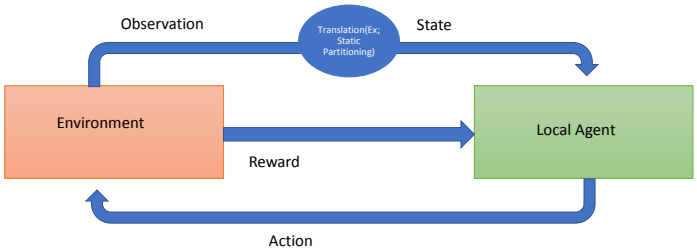


Figure 6.1: Main components of general reinforcement learning framework.

ing the violations of user level expectations. Therefore, the solution is targeting service level providers who have access to the VM resources and VM-level performance metrics.

6.4 Preliminary on Reinforcement Learning Framework

Figure 6.1 shows a general view of the RL framework for a problem which manifests the target environment. An agent is responsible to continuously monitor the environment and make observations of the important features. Collected observations are translated to one of the states s_i from the set $S = (s_1, s_2, ..., s_N)$. Each state represents an abstract description of the main features of the system. The final goal of RL is to gradually learn how to move between states to maximize a long-term objective function in terms of the total rewards from each action. Each movement is done by selecting an action a_i from the the set $A = (a_1, a_2, ..., a_M)$. At each decision time t, the agent decides to perform action a_t based on the obtained knowledge from previous movements and the current state s_t. The environment makes the requested changes based on the selected action and as a result the system may evolve to a new state s_{t+1}. The environment also sends back a scalar feedback (reward r_t) as the value of the action and its impact on the state of the system. These feedbacks are then used to update cumulative value of (s_t, a_t) pairs table which defines the goodness of selection a_t while in state s_t. The gradual learning happens as a result of the many trial and rewards in the form $S * A -> R$ over time to achieve an optimal policy for the agent.

Q-learning is an online, off-policy type of RL for continuous time Semi-Markov Decision Problems (SMDP) with the goal to obtain an optimal policy Π to maximize value

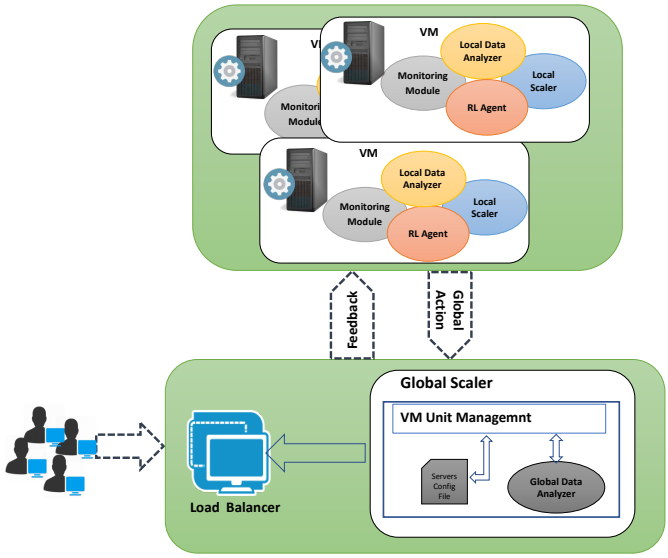


Figure 6.2: General Architecture of ADRL.

$Q_\Pi(s,a)$. $Q_\Pi(s,a)$ is a value function which estimates the accumulative discounted value of being in state s and performing action a under the policy Π which can be written as:

$$Q_\Pi(s,a) = \mathbb{E}_{s'}[r + \delta Q_\Pi(s^{'}, a^{'})|s,a] \tag{6.1}$$

where $\mathbb{E}$ is the expectation and $\delta \in [0,1]$ is the discount factor. It means that the value of performing action a in state s is the total amount of rewards that can be expected to accumulate by following policy Π from state s. Suppose that at decision epoch t action a_t is selected. At the next decision epoch $t+1$ and having the reward $r(s_t, a_t)$, the Q function value can be updated as follows:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha[r(s_t,a_t) + \delta max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)] \tag{6.2}$$

where $\alpha \in (0,1]$ is the learning rate. The greedy selection of action a' in the above equation without following the current policy, defines Q-learning as an off-policy approach as mentioned earlier.

6.5 System Design

Figure 6.2 depicts a high level view of main components of ADRL and their interactions with the external user and cloud environment. The users send their requests to the load balancer component which distributes them among existing active VMs. Figure 6.3 shows the details of 4 main modules in each VM as described in the following:

- *Monitoring Module* which is responsible for monitoring the measurable features of the environment. In the context of VM monitoring, these features can be resource utilization measurements such as CPU and memory.
- *Data Analyzer (DA)* performs data cleaning and behavior modeling of the VM. The aim is to create and continuously update an abstract model of VM performance and detect unexpected violations. The detected anomalies identify occurrence of performance problems and the need for corrective actions.
- *DRL Agent* is the main decision maker which is triggered after identifying an existing anomaly in the system by data analyzer module. It takes the observations from monitoring module of the system as input. The output of this module is an action that defines some changes in the configurations of resources. The selected action is fed to the local scaler or sent back to the global layer for further processing.
- *Local Scaler* is responsible for performing actions that define some type of the change in resource configurations of corresponding VM.

Algorithm 7 shows the main steps of ADRL framework. Each VM monitors the performance of its resources by collecting resource utilization metrics at regular time intervals. The collected data are fed into both local DA and RL agent for processing. DA utilizes feed-forward Neural Networks (NN) to perform a prediction of the future values of each collected metric. Then, the predicted values are used as input of an anomaly detection algorithm to decide if the system is behaving abnormal compared to the performance models from previous observation of VM. If an anomaly event is detected, DRL component is triggered to decide on a corrective action based on the observed state of the system. In this work, the performance anomaly detection is defined in favor of end users

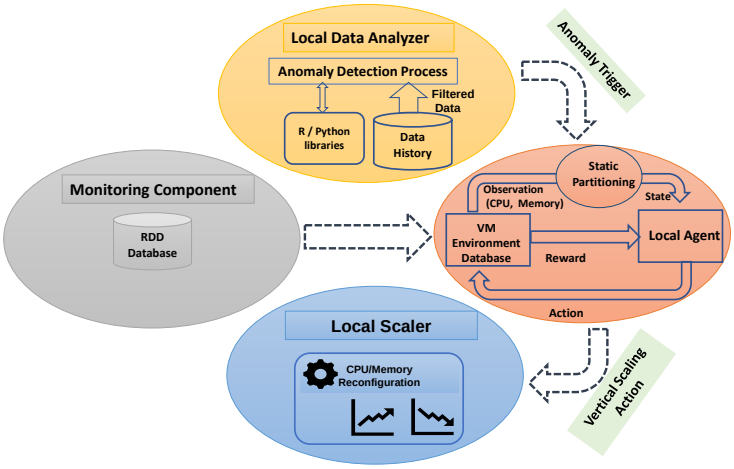


Figure 6.3: The Interaction among local ADRL components.

and points to the events that can possibly violate the expected Quality of Service (QoS) objectives. As a result, the anomaly event is defined as continuous and unusual changes in the values of VM performance metrics such as CPU and memory utilization which can affect the ability of the machine to process user requests in an acceptable time. Finally, it should be noted that DRL agent can also be triggered as a result of exceeding maximum *Time Between Actions (TBA)*. This condition is included for cases when the performance is in a normal state, but the resources are under-utilized. Although no anomaly is triggered during normal times, but we want to give the decision maker a chance to move toward states with higher utilization (possibly by removing extra resources).

Upon receiving the selected action from DRL, it is checked that the action is a local resource scaling request or not. If the answer is yes, the local scaler is called to adjust the amount of allocated resources based on the requested changes. On the other hand, if the action is a global scaling request, the results are sent back to the global scaler which is responsible for controlling the number of VMs. The global scaler can decide on adding new VMs to reduce the total amount of resource utilization in the system or shutting down the existing VMs to reduce under-utilized VMs and resource wastage. While the action is executing, the system enters a *Locked state* when no new action is performed. This strategy gives the system enough time to adapt to new configurations and reach a

stable state. The details of each step and corresponding algorithms are explained with more details in the following section.

Algorithm 7: ADRL: General Procedure

```
1  Initialize Q(s, a) table with historical transitions; Initialize anomaly detection
     Models;
2  while The system is running and in the beginning of performance-check interval do
       /* This part of the code is executed locally in each VM
          */
3      s_t ← Performance state for vm_i at time t based on the monitored data
4      if s_t shows an anomaly then
5          Increase the counter by 1;
6      end
7      if (counter ≥ L AND vm_i is not in Locked state) OR Time(a_{t-1}) ≥ TBA_max then
8          Call DRL Agent for a new Action a_t;
9          Execute a_t following Algorithm 8;
10         Schedule an update for learning model to be done according to Algorithm
             9;
11     end
12 end
```

6.6 ADRL: A Deep RL based Framework for Dynamic Scaling of Cloud Resources

In this section, we detail the main components of ADRL framework. As explained in Section 6.3 and Algorithm 2, ADRL is composed of three main parts to address the identified challenges in an adaptable resource management solution. We should note that this is a general architecture and each part can be easily extended to new data analysis techniques, more advanced resource management solutions such as migrations of VMs and other mapping techniques to select among state/action pairs. Table 6.2 presents a list of notations used in this chapter.

6.6.1 Deep Reinforcement Learning (DRL) Agent

The DRL module addresses the mappings of states to actions where a proper scaling action should be selected for current state of the system. Let us assume we have a pool

Algorithm 8: ADRL: Execution Phase

input : A_t: Selected action at time t

while *The system is running and in the beginning of performance-check interval* **do**
  `/* This part of the code is executed locally in each VM */`
  if A_t *is local* **then**
    Initialize all indicators in f to 0;
    for $a_j \in A_t$ **do**
      if a_j *is a request of change for resource* j **then**
        $R_j^{new} = R_j^{old} + a_j * R_j^{unit}$
        if $R_j^{min} \leq R_j^{new} \leq R_j^{max}$ **then**
          Apply the change
        end
      end
    end
  end
  else
    `/* This part of the code is executed in the master node */`
    Add new VMs or *Remove* from existing VMs based on the acceptable utilizations and state of the environemnt.
  end
end

Algorithm 9: ADRL: DRL Agent

`/* Select an Action */`
$s_t \leftarrow$ Performance state at time t based on monitored data
Choose an action from set A randomly with ϵ probability, otherwise select an action with maximum Q value;
`/* Perform scheduled learning */`
if *Learning schedule is triggered* **then**
  $s_{t+1} \leftarrow$ Performance state at time $t+1$;
  Calculate r_t based on Equation 6.6;
  Store stransition (s_t, a_t ,r_t, s_{t+1}) in VM profile memory M;
  Update Q according to Equation 6.4;
end

Table 6.2: Description for Notations

Notation	Description
R_j	Amount of resource j
R_j^{unit}	Unit of change for resource j. For example, one core for CPU resources
TBA_{max}	Maximum allowed time between actions
$V(s_t)$	Value of the state s_t
u_j	Utilization of resource j.
a_t	Action at time t
rt	Response Time
L	Minimum number of violations before the system reacts to an anomalous event

of active VMs $V = (v_1, v_2, ..., v_P)$ as our global environment. Each vm_i is described with a tuple $U = (u_{i1}, u_{i2}, ..., u_{iK})$ where u_{ij} is a scalar value representing the utilization of resource type j on vm_i. For each resource type j, an action a_j can be performed. If a_j is greater than zero, it corresponds to increasing resource j by amount a_j; If it is zero, it means the resource is unchanged and negative values correspond to amount of released resources. Therefore, depending on the total number of types of resources, the final set of the actions for each VM is defined as Cartesian product of the sub action sets of its resources as follows:

$A = \bigtimes_{j=1}^{K} a_j$

Accordingly, the purpose of DRL agent is to find a proper configuration of resources by continuing changes of respective resources and receiving feedback on the outcomes of the changes. However, the changes of resources on vm_i are limited to the minimum amount of allocated resources for a VM as well as the available resource of host machine. Suppose a scenario where the environment $V = (v_1, v_2)$ is handling the daily load of a web application with normal utilization of resources. The dynamic of web workload during the day is handled by adding/removing resources for each VM asyn-

chronously. Then, during a peak period, the load drastically increases which causes unexpected over-utilization of resources. In this scenario, the system is facing a situation that adding resources at local level may not be enough. Therefore, we add a special action a_{global} to the action set A where a_{global} corresponds to a request for help from global layer. Section 6.6.3 discusses these actions in more details.

DRL Agent $->$ Action Selection: Upon receiving an anomaly alert, DRL agent is called to choose an action in response to the detected performance problem. Let's assume that s_t is the observed state of the performance anomaly. In order to choose an action from the action set, we need a policy that exploits the available knowledge from feedbacks of previous decisions (exploitation) and also tests new actions to improve the knowledge of state/action relations (exploration). We use dynamic version of ϵ-greedy policy which is a standard policy for having a trade-off between exploration and exploitation policies. ϵ-greedy policy selects a random action with a probability equal to ϵ, otherwise it selects an action with Maximum Q value in the table. In order to have a dynamic policy with a higher exploration at the start, ϵ is initialized with 1 and as the number of observed states increases the value of ϵ decreases until it reaches a minimum value.

DRL Agent $->$ Learning-Model Update: When the system applies an action, a waiting time is required so the effect of changes can be reflected in the environment. At this time, DRL agent calls for an update based on the newly observed state s_{t+1}. The agent first stores the transition (s_t, a_t, r_t, s_{t+1}) in a profile memory. Then, the reward is calculated for the pair (s_t, a_t) to evaluate the goodness of the selection.

The final purpose of ADRL is to improve the QoS and utilization of services. Therefore, the reward is formulated according to this goal and is composed of three components as follows:

- *QoS*: The Quality of Service describes the level of satisfaction from user perspective. We choose response time (RT) as a measure for this metric. RT represents the waiting time for each request from submission to completion including runtime and queuing times. Let's rt be the average response time of requests during time interval t to $t+1$. Then, the reward of rt (R_{rt}) is calculated based on Equation 6.3 where RT^{max} and RT^{min} are maximum and minimum acceptable values. The

minimum value is considered to cover the cases when the VM moves to an unresponsive state and due to the limitations of resources no request can be accepted and as a result, RT drops to a near zero value.

$$R_{rt}\left(rt\right) = \begin{cases} e^{-(\frac{rt-RT^{max}}{RT^{max}})^2} & rt > RT^{max}, \\ e^{-(\frac{RT^{min}-rt}{RT^{min}})^2} & rt < RT^{min}, \\ 1 & otherwise \end{cases} \tag{6.3}$$

- *Resource Utilization*: While having an under-utilized environment can give the users a high QoS in terms of the running time of requests, the wastage of resources is not acceptable for service owners. The wasted resources increase costs in terms of the monetary value as well as energy wastage in the environment. Therefore, we need to consider the resource utilization for each resource j of vm_i in the final reward value. This value helps the decision maker to move toward decisions that increase the utilization of resources while considering the satisfaction of user expectations through QoS value introduced in the previous part. Equation 6.4 defines this value as an average of utilization on all resources where ${U_j}^{max}$ defines maximum acceptable utilization for corresponding resource j and $u_j \in (0, 1]$.

$$R_{ut}\left(u_j\right) = \begin{cases} \frac{\sum_{j=1}^{N} {U_j}^{max} - u_j}{N} + 1 & u_j \leq {U_j}^{max}, \\ \frac{\sum_{j=1}^{N} u_j - {U_j}^{max}}{N} + 1 & otherwise \end{cases} \tag{6.4}$$

- *State Transitions Value*: While running the experiments with ADRL, we noticed that a sequence of (s, a) transitions can lead the decision maker to be trapped in a loop between states. This can happen as a result of the simultaneous changes of resources by the actions that are affecting the value of more than one resource. This is especially important for applications where changes of one resource have a dominant effect in terms of the utilization compared to the others. Suppose we have vm_i with two resources CPU and memory in an under-utilized state. Action $a = \{-a, +a\}$ is triggered and one unit of CPU is removed while one unit of memory is added. Since the application is a CPU sensitive one, the utilization of CPU

significantly increases while memory shows a small change. Although the utilization of memory is still in under-utilized state, this action can result in a good reward value. Therefore, differentiating among transitions with utilization improvements of one resource can be challenging. Although this observation can be dependent on the units of changes and the characteristics of applications, considering the dynamicity and heterogeneity of cloud hosted applications this behavior can be expected. As a solution for this problem, ADRL introduces a state value function and transition penalty as Equation 6.5 where function V assigns manual weights to the states. If an action is causing a transition from a higher value state to lower ones, a penalty value is considered in the final reward function. In contrast, moving from lower state to higher states affects the reward value positively.

$$P\left(s_t, s_{t+1}\right) = \begin{cases} 1 & if V(s_t) < V(s_{t+1}), \\ -1 & if V(s_t) > V(s_{t+1}), \\ 0 & otherwise \end{cases} \tag{6.5}$$

Finally, Equation 6.6 shows the final value of $r(s_t, a_t)$ pair as the total rewards in terms of the QoS, utilization and state value changes. Higher values of R_{rt} and lower values of R_{ut} increase the final reward.

$$r(s_t, a_t) = \frac{R_{rt}\left(rt\right)}{R_{ut}\left(util\right)} + P\left(s_t, s_{t+1}\right) \tag{6.6}$$

Having all the information from transition (s_t, a_t, r_t, s_{t+1}) ready, updating of the Q-table can be done based on the new information and Equation 6.4. In order to improve the stability of learning and parameter updating in the presence of anomaly and temporal spikes which introduce abnormal transitions, we leverage experience replay as a sampling technique during training. This technique uses the profile of the past transitions to randomly select mini batches of records to be used for training the learning networks. Random selection of records also helps to overcome the correlation among sequential experiences as well as improving the efficiency by using each experience in many of the updates [169].

6.6.2 Anomaly-aware Decision Making

In the context of cloud resource management, the actions are triggered as a response to the performance problems in the system. However, the base DRL loop usually works as a periodic decision maker with iterative selection and updating steps to gradually adapt to the environment. Proactive event-based decision making is another approach where the decisions are made as a response to possible predicted performance problems. This helps the system to reduce the frequency of decision makings which also reduces the possibility of oscillation among states. In order to achieve this goal, we choose IForest technique as described in Chapter 3. IForest model is built based on an ensemble of many iTrees and the anomaly scores are average of path length on all trees. Having a worst time and space complexity $O(T\psi^2)$ and $O(T\psi)$ for training of T iTrees, it is a promising option for dynamic environments where the models require regular updates to capture the latest state of the system.

One point worth mentioning here is that the triggering of an anomaly state can be a result of a change between states in terms of the values of monitored metrics from workloads and VMs. Three problems arise as a result of this transition to be addressed.

First, the transitions among states can be a result of temporal spikes which can be expected in highly dynamic environments. To address this problem, one anomaly alert is not taken as a serious anomaly event. In fact, DRL agent is triggered for making a decision when a continuous anomaly event is identified by receiving at least L consecutive alerts (Algorithm 2, Lines 4-7). Therefore, the system ignores the first few alerts to avoid unnecessary reactions to transient changes. The value of L can be decided based on a combination of factors such as system logging interval, application characteristics and the degree of fault tolerance.

Second, if the transitions are real, the trained anomaly detection models may not reflect the new states and therefore there will be many false anomaly alerts. To solve this problem, we use the same idea introduced in Chapter 5 for deciding the proper time for updating of the models. In our case, an update will happen when the transition is completed and therefore the new observations are representing new behavior of the system.

Finally, it should be noted that while the frequency of decision making is reduced

by replacing the periodical triggering with anomaly triggers, we should still consider that not all decision epochs require a change in the states. If the performance is in a good state in terms of the reward values, no-change actions may give a better chance of reaching an optimal condition. Action a_j equal to zero as discussed in section 6.6.1 helps the system to experience the no-change effect on the performance of VMs.

6.6.3 Two-level Scaling

As we explained in Section 6.6.1, two levels of scaling are considered in this work. The first level is defined for each resource of VMs. Three types of the action as defined by a_j are applied based on the units of change for each resource. Let us assume one CPU core as the unit of the change for this resource. Therefore, $+a$ action increases the number of cores by a while $-a$ action removes a cores from the VM. Similarly, the unit of changes for memory can be set as $256MB$ and therefore each action changes the amount of allocated memory with multiples of this unit. In our work, one unit is selected for each change. Moreover, an action is valid if the requested changes are not violating the available resource of host machine or minimum acceptable amount of the allocatable resource to each VM.

The second level of scaling is at global level which is responsible for managing the units of VMs and can change the number of VMs according to the state of the system. Therefore, global scaler should have access to the utilization of all VMs. ADRL designs global layer as a threshold based horizontal scaling algorithm. In an under-utilized environment, the global scaler identifies the VMs which have lower utilization of an acceptable minimum threshold and shut-down or deactivate these machines. Similarly, when the scaler finds environment in an over-utilized state, new VMs are added to help reduce the load on existing machines.

6.7 Performance Evaluation

In this section, the performance of the proposed framework is evaluated using CloudSim discrete event simulator [163]. An extension of CloudSim is used that includes analytical

performance models of a web application benchmark [164] and an anomaly injection module. The simulator helps us to create a controlled environment for performance anomaly testing and corresponding validations for different types of problems.

6.7.1 Experimental Settings

We model the environment as a data center with two types of application and database server VMs. The configuration of VM templates for application server is one virtual core, 256 MB of Ram and Linux operating system and the maximum limit for resources are 5 cores and 3072 MB, respectively. The workloads are based on the web-based user requests on Rice University Bidding System (RUBiS) benchmarking environment which models an auction site following ebay.com model. Each session of the web workloads is modeled based on the monitored resource usages of real requests on RUBIS [164]. To generate the performance models of system, four attributes CPU, memory, disk utilization and number of sessions are collected. VM start-up times are also modeled based on the study done in [119].

The anomaly detection module is initialized by generating iTrees models for each individual VM. Unless otherwise specified, the value of parameters in IForest configurations and model updating schedules are according to the recommended settings as explained in Chapter 5. The value of L is set equal to 6 based on the logging intervals and the characteristics of the application.

In order to initialize Q-table of DRL agent, we run CloudSim for 48 hours and record the transitions and corresponding rewards in a file. These records are then used in a batch learning process to initialize the Q values [169]. For Deep Q-learning we use a constant learning rate $\alpha = 0.05$ value and a discount factor $\gamma = 0.9$. The number of layers is 20 and the size of mini-batches for profile memory is 50 based on our experimental evaluations. ϵ is decreased from 1 to 0.1 which gives higher exploration capability in the initial iterations of learning with ϵ-greedy policy.

In order to assign weights to states for penalizing process, we follow a simple idea based on the static partitioning of the state space. Therefore, for each resource, the utilization is divided to 5 partitions and the incoming state values are mapped to the cor-

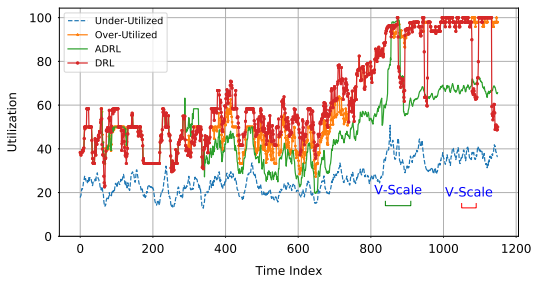
responding partition. Partitions with higher utilization get higher weights. DRL agent is implemented in Python environment with TensorFlow and a wrapper is created to connect Java-based CloudSim simulator to python codes.

Each experiment has a duration of about 24 to 48 hours. The normal workload is based on RUBiS benchmark and the sessions are generated based on Poisson distribution with a time-based frequency as explained in [164]. Two types of CPU and memory anomalies are generated in CloudSim to create an increasing trend effect in the consumption of CPU and memory without significant changes in the normal load of the system. These anomalies start after the model initializations and at random times during execution. To create the increasing load effect, after 10 hours of normal load, the number of sessions start to increase in two phases by adding 5 and 20 sessions at each time unit, respectively.

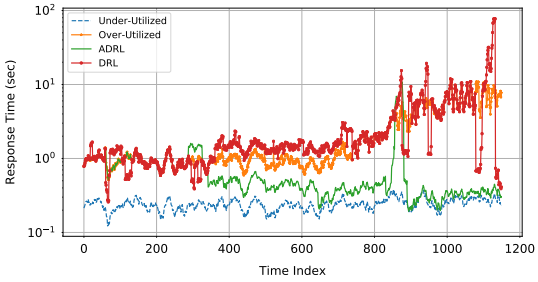
6.7.2 Experiments and Results

In order to evaluate the performance of ADRL, two static methods and one DRL based method are considered. In *Under-Utilized* method, the VMs are configured so that the total amount of allocated resources is more than the demanded ones. Therefore, with an under-utilized method, the user can experience the best QoS. In *Over-Utilized* case, the VMs are set up based on the minimum VM template configurations as described in Section 6.7.1 such that during the run of the experiment and by starting anomaly events the utilization of resources exceeds the acceptable level and some violations are allowed. In both cases, no scaling is done through the experiments, therefore generating a sample of the best and worst results to evaluate the general functionality of ADRL. We also implement a non-anomaly aware RL based algorithm similar to works such as [64]. To have a fair comparison, we extend their RL implementation with Deep Learning Decision maker and hybrid of vertical and horizontal scaling actions and name it as DRL to study the effect of anomaly based decision making of ADRL.

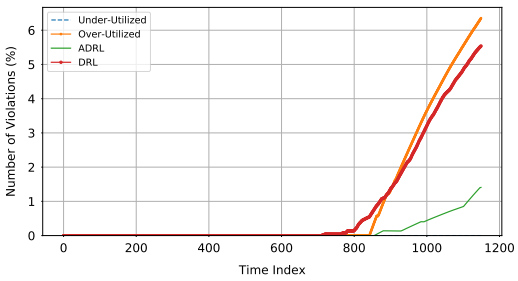
Figure 6.4 presents the results of all methods on a workload with CPU hog problem. The first diagram shows the CPU utilization corresponding to each scenario. As we can see, under-utilized environment shows the lowest CPU utilization while over-utilized



(a) CPU Utilization

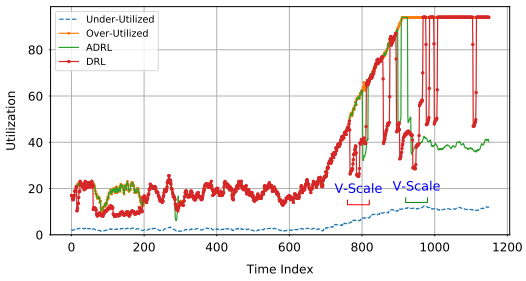


(b) Response Time

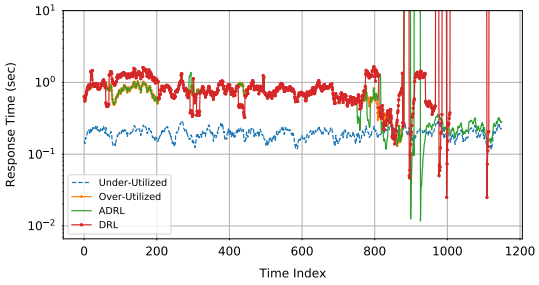


(c) SLA violations for CPU anomaly

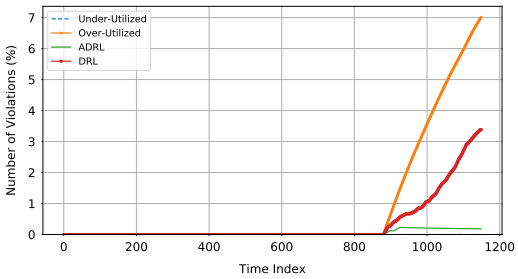
Figure 6.4: CPU Utilization, Response Time (Log) and violations number for CPU shortage dataset. ADRL is able to pro-actively trigger vertical scaling actions in response to anomaly events (utilization more than %80). It also shows higher stability in comparison to DRL with multiple changes of state between anomalous and normal states



(a) Memory Utilization



(b) Response Time. Vertical upward spikes show the violations of SLA in terms of response time. ADRL can eefectively decrease the number of violations by desciding to add more resources and keep the system in stable state while time-based descion making by DRL is returning back the system to an anomolous state by constant moves among states.



(c) Total percentage of violations. As the graph shows, with the start of anomaly and violations of SLA, ADRL adds extra resources which avoids further increases in the failed sessions.

Figure 6.5: Memory Utilization, Response Time and cumulative violations in the presence of memory shortage dataset. ADRL is able to pro-actively trigger vertical scaling actions in the response to anomaly alerts which decreases RT violations and rejected sessions.

one has the highest utilization. While CPU consumption is increasing, both DRL and ADRL try different types of actions. These actions are not always the optimal choices which is expectable as the system is observing new states that may have a few history records of their transitions before. However, as the system starts to violate the QoS around $t = 800$, both algorithms try to reduce the utilization by adding new cores to the VM. At this point, ADRL observes a transition in the utilization values, updates the anomaly detection model and enters a stable state. The stability of process can be seen around observation $t = 900$ and onward where no anomaly is triggered and therefore no action is performed to change the states. In contrast, DRL continues time-based decision making which may return the system back to the violation state. Although choosing $a_j = 0$ action can help the system to keep the current state, but some actions which are resulted from random selections or due to the temporal spikes of the performance can cause wrong changes of configurations and extra violations. These violations are also shown in the last graph of Figure 6.4. This diagram shows the cumulative percentage of violations during each time interval. As the picture shows, ADRL can reduce the incremental results of QoS violations in the presence of anomalous behavior by performing vertical scalings and keeping the system in normal state. In contrast, DRl can not show stable results in terms of violation reductions as it contentiously returns the system back to abnormal state. As already mentioned, these behavior is due to not recognizing the continuity of anomaly state and trying to make new changes to maximize rewards with regard to resource utilization.

Figure 6.5 shows the utilization and RT diagrams for memory shortage problem in the system. To generate the anomaly state, after $t = 600$, a steady increase of the memory utilization is started and the results of each scenario for memory utilization and RT are presented. The diagrams for under-utilized scenario do not show any significant change as there is still plenty of free memory available. In contrast, over-utilized execution gets affected immediately as the utilization exceeds corresponding thresholds which are reflected in the second diagram where RT shows sudden increases. These unexpected increases which are shown as vertical upward lines in the graph happen when the VM does not have enough memory and therefore becomes unresponsive while rejecting many of the new incoming requests. However, with the start of memory anomaly

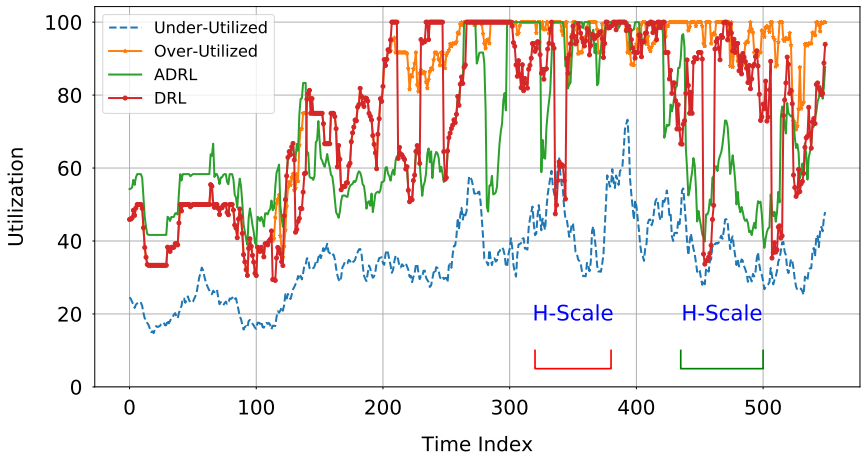


Figure 6.6: A combination of vertical and horizontal scaling actions in overloaded system. Two scaling actions done by ADRL and DRL methods are shown as an example

and increase in RT violations, ADRL decided to add extra resources which avoid further violations as well as decrease the number of failed sessions. DRL, in contrast, achieves an initial decrease of RT violations by adding more resources; however, the time based triggering of decisions and sudden spikes of utilization while moving between states cause wrong actions which release some of the resources. The sequence of these add/removal of resources causes several violation spikes and returning the system back to the anomaly state. This is again due to the ignoring of the stability of system in terms of being in an identified continuous anomalous state and particularly is expected when the system is experiencing higher explorations. For example, this can happen when the system is observing rarely seen states such as memory utilization higher than %30 in a CPU-intensive application. ADRL, however, correctly identifies anomaly states and after two wrong configurations, around $800 \leq t \leq 900$ brings the system back to a steady performance. The last diagram of Figure 6.5 demonstrates the results of cumulative number of violations which highlights the ability of ADRL to reduce the total number of violations after detecting the anomalous behavior with regard to the memory utilizations.

In order to show the response of the system to high load problems and triggering of horizontal scaling actions, we run CloudSim with a workload that increases the load to saturate resources. Figure 6.6 shows the corresponding CPU utilization of this load and

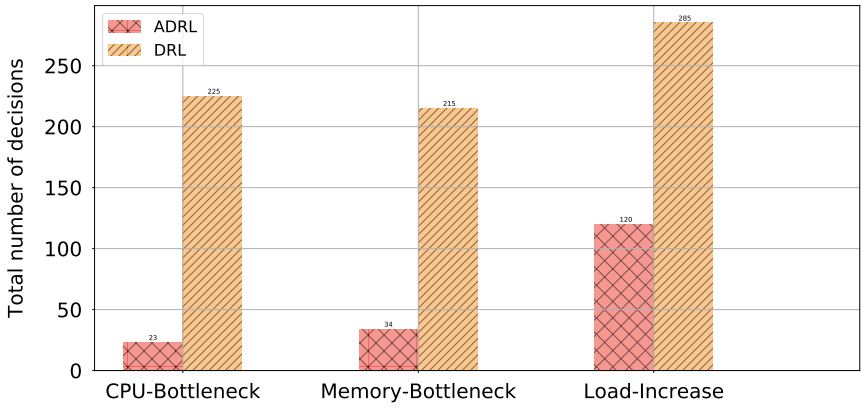


Figure 6.7: Total number of decisions (scaling actions) for both methods DRL and ADRL for each dataset. ADRL is able to decrease the number of decisions with an event-based decision making process.

the changes made in the system for static and dynamic scenarios.

As we expect, the under-utilized run shows the lowest utilization, while the over-utilized configuration soon reaches the saturation point of resources. Both DRL and ADRL trigger a mix of vertical and horizontal scalings during their run. The horizontal scaling decisions that add new VMs for DRL and ADRL are shown with red and green marks on the diagram, respectively. However, the sequence of decisions made by DRL during the transitions of system from abnormal to normal state weakness the expected effects of added VM in the system. The reason is due to the decisions that remove some cores from existing VMs which can temporally reflect increases of the utilization. However, the increase is happening during the transition of system when the load is still increasing which as a result causes the violations of performance. In contrast, ADRL correctly identifies the continuous anomaly events and the number of decisions in the presence of temporal spikes is less and more accurate.

Figure 6.7 shows the number of decisions corresponding to the scaling actions for both methods ADRL and DRL. As we have mentioned before, DRl includes a periodic decision maker while ADRL triggers scaling actions in the response of detected anomalies. As a result, ADRL can significantly decrease the number of scaling actions. This reduction is important in cloud environment as every scaling is changing the patterns of

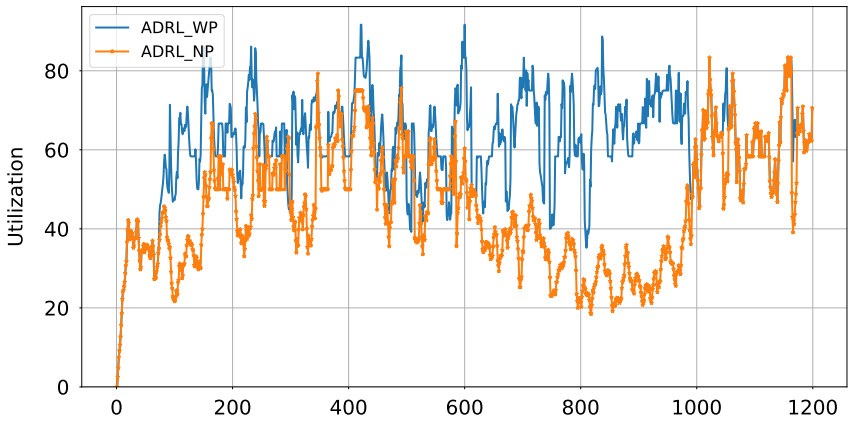


Figure 6.8: A comparison of CPU utilization with two versions of ADRL. ADRL_WP performs penalizing process as part of the reward calculation while ADRL_NP ignores this step.

the performance in the system and therefore affecting the accuracy and updating interval of prediction models.

Finally, to validate the effect of penalty values of the reward function (Equation 6.6) in guiding the decision maker to higher value states, we run two versions of ADRL with penalties included (ADRL_WP) and without that (ADRL_NP). The results of this experiment are shown in Figure 6.8. As we can see, ADRL_NP selects more action types that increase resource allocations and moves the system to the states with lower utilization which as described in Section 6.6.1 have lower value in accordance with the reward function. For example, there are a series of decisions to add resources around $t = 300$ or between $t = 600$ to $t = 900$ which reduces the utilization. However, by each reduction, the utilization part of the reward function reflects the negative effect of these movements which helps the system to recover (as it is shown around $t = 1000$) after a few steps. However, ADRL punishes the decisions that move the system to low utilization states while encouraging toward decisions that remove resources when the utilizations have not reached their maximum thresholds. Therefore, the general behavior of the system under ADRL management is more toward high utilization states with higher values as long as the SLAs are respected. This helps the system to quickly learn about the actions which configure resources to achieve higher reward values.

6.8 Summary

In this chapter, ADRL is proposed as a two-level adaptable resource scaling framework. ADRL models the problem of resource scaling as a Deep Reinforcement Learning (DRL) framework with the capability of observing the performance of surroundings and taking actions as a response to the problems. ADRL identifies performance problems by using an anomaly detection model and the actions are combinations of horizontal and vertical scaling changes. The anomaly detection model helps to identify continuous performance problems. DRL agent is triggered based on the detected event and tries to find proper scaling actions which maximize a reward function defined in terms of the QoS and resource utilization.

We also proposed a penalizing mechanism to guide the DRL decision maker toward the actions that move the system to higher value states. Through an extensive set of experiments, we show that ADRL framework can achieve better results in terms of identifying and correcting the performance problems with a smaller number of decisions. Moreover, it is shown that different types of performance anomalies can be addressed by scaling decisions at various levels of granularity.

Chapter 7
Conclusions and Future Directions

This chapter concludes the thesis and discusses a summary of works and key contributions in this research work. Then, it discusses some of the identified challenges and future works for performance-aware resource management in cloud.

7.1 Conclusions

The virtualization and elasticity feature of cloud resources has brought up many opportunities for on-demand sharing of distributed resources. The resource manager controls the amount of resources in the system with regard to the amount of workload and expected QoS for individual applications. The violations of QoS and SLA agreements can cause cloud providers monetary costs and their reputations. However, the highly dynamic environment of cloud, comprising of dynamic workloads, distributed resources with possible hardware-level faults, software-level bugs or resource sharing conflicts can make the performance of the system highly unstable and unpredictable. This highlights the need for advanced resource management solutions which are aware of the performance of the system and can interact with environment to identify the problems at different levels of granularity.

On the other hand, the advances in monitoring and data analysis techniques provide a valuable source of processable information to track the performance of the system and applications with the aim of finding the preliminary signs of problems to act upon in terms of adjusting the configurations of allocated resources. This thesis investigated the joint performance analysis and resource management frameworks where the former part tries to detect the possible performance problems while the latter leverages this

knowledge for improving their resource allocation decision making.

Chapter 1 presented the background and main research questions and contributions with regard to adaptive performance-aware resource management in this thesis. Then, Chapter 2 discussed these terms in more details and proposed a taxonomy for categorizing the existing literature in the area of performance dependent resource management. This chapter surveys related works in each category and compares their main contributions with regard to the applied data analysis approach or resource management techniques. The survey of current literature also assisted to understand the current gaps and open research questions which some of them were investigated in this thesis.

Chapter 3 presented an anomaly detection process based on time-series pre-processing techniques and isolation-tree (iTree) data structures. To show the effectiveness of the approach in terms of the correctness and precision, several web-based workload datasets were generated by deploying a benchmark in a private cloud environment. The deployed system included main components of a web-application including web and database servers. Various types of the anomalies such as CPU and memory bottlenecks were injected and the final datasets were collected by monitoring the performance of the components during the running of the system. These datasets were time-series of the utilizations and workload attributes which jointly create an abstract representation of the performance of the system. Efficacy of the solution was validated by two metrics AUC and PRAUC for different datasets in the presence of the performance problems.

Chapter 4 targeted the isolation-based anomaly detection problem for high-dimensional data by leveraging the knowledge from iTree data structure to filter irrelevant and noisy features. This process makes the isolation-based anomaly detection much faster in terms of the modeling and testing times. The process is performed by targeting the features that isolate anomaly instances in short branches of iTree. According to the definition of anomalies as being rare and different, these features are supposed to have higher contribution in detecting anomalous records. The process, then, is validated on several benchmark datasets to show that the reduced features can improve the detection results while significantly reducing the training times by reducing the number of features and iTrees.

Upon the development of anomaly detection module, a joint performance anomaly

analysis and resource scaling decision maker is proposed in Chapter 5. The main purpose of this chapter is to show how the proactive identification of problems can help the decision maker to select proper scaling action type for alleviating the performance degradations. The proposed framework (ACAS) has been implemented in the extension of CloudSim which is a discrete event-based simulator for cloud based systems. The anomaly detection module acts as the trigger of decision maker which is called upon receiving an alert of possible problems in the system. Two levels of the problem, local for VM specific resource related problems and global for system related load problems are investigated. A simple cause inference module identifies the source of the problem. Then, local problems are resolved by vertical solutions while global load problems are responded by horizontal level scaling solutions. Moreover, a new model updating algorithm is proposed to identify the times that the anomaly detection models require new training with recently observed data. ACAS has been shown to effectively respond to CPU and memory problem with proper vertical resource changes in comparison to conventional solutions that target this type of problem with the same horizontal level solutions and adding/removing VMs which is more time-consuming. Load problems also take the advantage of ACAS proactive decision making which gives the resource manager enough time to add new VMs before the degradations in performance violates the expected QoS.

While ACAS shows the advantage of using performance anomaly information in improving the quality of resource decision maker, the mapping between performance problem and action type is pre-decided with a series of threshold based if/else rules. However, considering the limitation of available resources and dynamicity of state of the system in terms of the various utilization metrics and corresponding performance problems, an adaptive solution that can interact with and learn from the environment is preferred. Therefore, in Chapter 6, we extended our anomaly triggered resource scaling framework with a Reinforcement Learning (RL) architecture. Both scaling types are encoded as action set of RL while the state space is comprised of utilization of resources. To overcome the dimensionality problem of state space, two strategies are considered. First, the distributed implementation of the framework which allows each VM monitors its own sate and as a result training/updating of performance anomaly detection and

RL models can be done locally. Second, deep neural nets are used to approximate the relation between state/action space and expected reward from the environment. The joint of these strategies makes the final framework scalable and effective in handling local and global anomaly problems. Moreover, the proposed solution can achieve high adaptivity as it is shown in handling various types of local and global performance problems.

7.2 Future Directions

The research in this thesis contributes to some of the challenges in joint anomaly aware computing resource management in the cloud. However, there are still other aspects on both sides of the performance data analysis and resource management to be investigated more comprehensively. This section gives some insights into these challenges for future work in this area.

7.2.1 Supporting Resource-limited Computing Units

With the advancements in Internet of Things (IOT) devices and their interconnection with cloud hosted recourses, the definition of units of computing is extending from VM and containers to smaller, resource constrained devices such as wearable and smart appliances. Although these devices are usually connected to another layers of computing such as edge and cloud resources, they may still require a level of processing functionality for in-site analysis of information. Therefore, a new category of customized solutions is required for both parts of performance analysis and resource management. With regard to data analysis part, limitations of resources restrict the applicability of complex analysis and require fast, memory-efficient solutions. A solution might be having a hierarchy of analysis where the preliminary processing is done in the device and further in-depth analysis is requested to be done by more powerful connected computing layers. Similarly, the resource management decisions are impacted by the limitation of resources where an efficient load balancing and offloading among connected devices and cloud hosted computing resources should be done. Finally, depending on the type

of the application and their requirements, a reformulation of QoS parameters and SLA definitions may also be required. For example, a health-related application on a resource limited device creates a need for high precision data analysis algorithms with low false alarms to have more efficient utilization of available resources.

7.2.2 Energy Efficiency

The flexibility of selecting among abundant resources on an on-demand basis and virtual view of infinity of resources comes with the cost of thousands of servers running and consuming an enormous amount of electricity. The cost associated with this energy consumption encourages resource providers to find more efficient solutions in terms of the energy usage while taking into account the expected performance of their services into account.

The joint management of performance and energy requires a deeper understanding of the workload patterns and more advanced fault tolerance strategies. In the context of cloud resource management, having a history of application resource usage, profiling of performance on various configurations and understanding of performance degradations from resource contentions are part of the knowledge to be learned for better decision making. Moreover, the availability of new sources of clean energy such as wind and solar introduces new opportunities and challenges for offering more efficient resource utilization solutions.

7.2.3 Adaptable Learning in Cloud

We have already proposed gradual autonomous learning frameworks such as RL as a solution for having more adaptable resource management. This area is rapidly growing with many promising techniques for improving learning efficiency. Deep Q-learning (DQN) networks are an example of these techniques. However, there are other strategies to further improve the training convergence and adaptability for possible scenarios. For example, in the context of resource management, we introduced *state weighting* and *no-change* action to customize the learning for the states with higher values. Another version of DQN [170], Dueling DQN, targets this problem by splitting the state value

from action values. Therefore, a state can have its own value without consideration of applied action. In theory, this should help to recognize the states which are valuable (or not), no matter what type of the action is selected.

Moreover, considering the potential of DRL frameworks to process high volume of data, it will be interesting to investigate the effect of integrating anomaly related information, such as anomaly scores for a variety of metrics to the definition of the state. Considering the direct relation between the degree of anomalousness of a metric and corresponding vertical scaling solutions, this information may further improve the quality of the decision making process.

7.2.4 Cause-aware Performance Data Analysis

Performance degradations can happen as a result of low-level hardware faults to high-level user based malicious attacks and etc. Current literature, as discussed in Chapter 2, conventionally investigates these problems separately by targeting various system/application attributes at different levels of granularity. However, the interdependency of components causes the propagation of problems which results in many correlated problems at different layers of computing environment. For example, a malicious network attack triggers scaling of resources by simulating a high load performance problem in an over-utilized system. However, in this context, new resources increases the cost as well as energy consumption for fake users that should not contribute to the performance evaluations of the system. Having an integrated approach is required so the performance analyzer can track down the source of the problem and make a decision according to the identified cause. In this case, a pre-knowledge of component dependencies at application level as well as access to different levels of information from network packet data to operating system calls and resource-level utilizations is a challenge to be investigated more. One way to achieve this is an agent based approach where the interaction among agents disseminates information about unique problems at different levels of granularity. While this strategy offers greater levels of scalability, the communication protocol, synchronization and consistency of information or the speed of information spreading are among added overheads to be considered for a highly distributed solu-

tion.

7.2.5 Customized VM Configurations

Public cloud providers such as Google cloud [5] offer a possibility to request for VMs with custom hardware settings. Considering the heterogeneity of cloud based applications with different levels of CPU and memory requirements, the knowledge from performance analysis techniques can help to better identify the exact VM templates which can satisfy resource requirements of application while considering the cost of resources and energy consumption. Traditional horizontal solutions usually consider a homogeneous VM environment to further simplify the target problem. However, this approach might not be well suited to heterogeneous environments where the choice of VM template can directly impact the performance and future resource requirements, particularly in terms of the energy and cost metrics. Therefore, initial VM configuration can be considered as another variable to have more adaptable resource management solutions which suit heterogeneous types of applications.

7.2.6 Application-aware Scaling Strategies

Considering the level of heterogeneity in cloud systems, a wide variety of applications and data can be hosted and stored on the VMs. However, not all applications have horizontal scaling capability, meaning that the duplicates of the service are not possible. The lack of support for scalability comes from a variety of reasons such as vendor locked-in, architectural limitations such as database syncing problems, sticky sessions for web applications and etc. Moreover, legal and security related issues also can limit the horizontal scaling options when there are strict requirements on the placement and migration of data and applications. Therefore, having a detailed knowledge of the application characteristics might be necessary as another level of information to improve the applicability of scaling decisions in these systems. The knowledge can be added as new constraints for decision maker to adapt their actions to these requirements. This can affect the decisions on the location of new VMs, the maximum number of service duplicates and as a result the highest load that can be handled, VM migrations and

consolidations. For example, for a database without the syncing capability, vertical solutions may be the only option for the scalability of application.

7.2.7 Performance-aware Advanced Reservation

With the advances in big data analysis techniques and availability of large volumes of data with higher quality in terms of the details and accuracy, precise resource utilization prediction and analysis are possible. Particularly, long-term predictions can be done by analyzing the regular patterns, seasonality and trends of data in long-run. This analysis gives service administrators better understanding of future usages and an insight on time-dependent performance bottlenecks and degradations.

On the other hand, admission and reservation based resource management mechanism has been used extensively in literature to ensure the QoS requirements of specific applications. An efficient reservation mechanism helps to pro-actively plan for resource reconfigurations based on the expected variations in the workload, application-specific updates which changes pattern of usage, peak times and etc. The effectiveness of designed plans highly depends on the amount of historical data available, the quality of predictions and the probability of sudden unexpected anomalous events. Sudden performance degradations may not be captured by prediction techniques and still require reactive mechanisms to be corrected. However, a variety of anomaly detection mechanism and deep learning solutions helps to get the highest level of knowledge on short and long term events for planning the predictable part of the performance profiles. For example, a short-term prediction of an anomalous memory leak event can help the system to pro-actively reserve extra memory on the hosted machines to be added to the VM when the QoS gets close to the threshold values. Combining these techniques help to further improve the reliability of the system in terms of avoiding and handling performance degradations.

7.2.8 Considering Specific Workload Requirements

While the proposed approaches in this work are general and can be customized to a variety of requirements, there are cases where these methodologies need some extension

to be properly functional. For example, to manage streaming workloads with hard real-time requirements, we need extra information on the constraints to create trade-off between the efficiency of the proposed solutions and meeting the constraints. We may need to combine a variety of approaches such as reserved resources, more sensitive anomaly detection thresholds or using the voting mechanism and multiple anomaly detectors. As another example, workloads with pattern based anomalous behavior may not be efficiently managed with the proposed anomaly detection approach as the main assumption in our methodology is that anomalies happen as a result of unexpected changes in the point values. Therefore, to efficiently process and detect anomalies for these types of workloads, we need to take extra steps to define the patterns of normal data and detect anomalous clusters of data.

Bibliography

[1] Flexera, "Rightscale," Tech. Rep. [Online]. Available: https://info.flexerasoftware.com/SLO-WP-State-of-the-Cloud-2019

[2] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[3] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud computing: Principles and paradigms*. John Wiley & Sons, 2010, vol. 87.

[4] Amazon, "Amazon," 2019. [Online]. Available: https://aws.amazon.com/

[5] Google, "Google cloud," 2019. [Online]. Available: https://cloud.google.com

[6] B, "News," accessed: 2019-05-23. [Online]. Available: https://www.evolven.com/blog/2011-devastating-outages-major-brands.html

[7] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Computing Surveys*, vol. 48, no. 1, pp. 4:1–4:35, Jul. 2015.

[8] B. Subraya, *Integrated Approach to Web Performance Testing: A Practitioner's Guide*. IGI Global, 2006.

[9] ProfitBricks, "Profitbricks," 2019. [Online]. Available: https://www.profitbricks.com/help/Live_Vertical_Scaling

[10] M. Azure, "Azure," 2019. [Online]. Available: https://azure.microsoft.com

[11] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, pp. 1–6, 2006.

[12] Ganglia, "Ganglia," 2019. [Online]. Available: http://ganglia.sourceforge.net/

[13] A. Anwar, A. Sailer, A. Kochut, and A. R. Butt, "Anatomy of cloud monitoring and metering: A case study and open problems," in *Proceedings of the 6th Asia-Pacific Workshop on Systems*. New York, NY, USA: ACM, 2015, pp. 6:1–6:7.

[14] H. J. Syed, A. Gani, R. W. Ahmad, M. K. Khan, and A. I. A. Ahmed, "Cloud monitoring: A review, taxonomy, and open research issues," *Journal of Network and Computer Applications*, vol. 98, pp. 11 – 26, 2017.

[15] S. Di, D. Kondo, and W. Cirne, "Google hostload prediction based on bayesian model with optimized feature combination," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1820–1832, 2014.

[16] K. Cetinski and M. B. Juric, "Ame-wpc: Advanced model for efficient workload prediction in the cloud," *Journal of Network and Computer Applications*, vol. 55, pp. 191–201, 2015.

[17] L. Yazdanov and C. Fetzer, "Vscaler: Autonomic virtual machine scaling," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. IEEE Computer Society, 2013, pp. 212–219.

[18] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, "Workload predicting-based automatic scaling in service clouds," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 810–815.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[20] 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*, ser. SRDS '13. Braga, Portugal: IEEE Computer Society, 2013, pp. 205–214.

[21] C. A. Cunha and L. Moura e Silva, "Separating performance anomalies from workload-explained failures in streaming servers," in *Proceedings of the 12th*

IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 292–299.

[22] A. B. Ashfaq, S. Rizvi, M. Javed, S. A. Khayam, M. Q. Ali, and E. Al-Shaer, "Information theoretic feature space slicing for statistical anomaly detection," *Journal of Network and Computer Applications*, vol. 41, pp. 473–487, 2014.

[23] J. . Cid-Fuentes, C. Szabo, and K. Falkner, "Online behavior identification in distributed systems," in *34th IEEE Symposium on Reliable Distributed Systems*, Montreal, Quebec, Canada, Sept 2015, pp. 202–211.

[24] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing–degrees, models, and applications," *ACM Computing Surveys(CSUR)*, vol. 40, no. 3, pp. 7:1–7:28, 2008.

[25] X. Gu and H. Wang, "Online anomaly prediction for robust cluster systems," in *Proceedings of the 25th IEEE International Conference on Data Engineering, 2009*. Shanghai, China: IEEE, 2009, pp. 1000–1011.

[26] 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*, Macau, China, June 2012, pp. 285–294.

[27] 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*. New York, NY, USA: ACM, 2012, pp. 191–200.

[28] 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 Trans. on Systems, Man, & Cybernetics: Systems*, vol. 46, no. 1, pp. 61–75, 2016.

[29] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vperfguard: An automated model-driven framework for application performance diagnosis in consolidated cloud environ-

ments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 271–282.

[30] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "Fchain: Toward black-box online fault localization for cloud systems," in *Proc. of the 33rd IEEE Intl. Conf. on Distributed Computing Systems*. IEEE, 2013, pp. 21–30.

[31] H. Nguyen, Y. Tan, and X. Gu, "Pal: Propagation-aware anomaly localization for cloud hosted distributed applications," in *Proceedings of the Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, ser. SLAML '11. New York, NY, USA: ACM, 2011, pp. 1:1–1:8.

[32] D. J. Dean, H. Nguyen, P. Wang, and X. Gu, "Perfcompass: Toward runtime performance anomaly fault localization for infrastructure-as-a-service clouds," in *Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 16–16.

[33] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang, "Perfscope: Practical online server performance bug inference in production cloud computing infrastructures," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 8:1–8:13.

[34] J. Tucek, S. Lu, C. Huang, S. Xanthos, and Y. Zhou, "Triage: Diagnosing production run failures at the user's site," in *Proceedings of 21st ACM Symposium on Operating Systems Principles*, 2007, pp. 131–144.

[35] X. Zhang, F. Meng, P. Chen, and J. Xu, "Taskinsight: A fine-grained performance anomaly detection and problem locating system," in *Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing*, June 2016, pp. 917–920.

[36] T. Do, M. Hao, T. Leesatapornwongsa, T. Patana-anake, and H. S. Gunawi, "Limplock: Understanding the impact of limpware on scale-out cloud systems," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 14:1–14:14.

[37] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 193–204.

[38] Y. Zhuang, E. Gessiou, S. Portzer, F. Fund, M. Muhammad, I. Beschastnikh, and J. Cappos, "Netcheck: Network diagnoses from blackbox traces," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 115–128.

[39] W.-C. Lin, S.-W. Ke, and C.-F. Tsai, "Cann: An intrusion detection system based on combining cluster centers and nearest neighbors," *Knowledge-Based Systems*, vol. 78, pp. 13 – 21, 2015.

[40] M. A. Hatef, V. Shaker, M. R. Jabbarpour, J. Jung, and H. Zarrabi, "Hidcc: A hybrid intrusion detection approach in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 3, 2018.

[41] P. Shamsolmoali and M. Zareapoor, "Statistical-based filtering system against ddos attacks in cloud computing," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, Sept 2014, pp. 1234–1239.

[42] C. Qu, R. N. Calheiros, and R. Buyya, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances," *Journal of Network and Computer Applications*, vol. 65, pp. 167–180, Apr. 2016.

[43] V. Persico, D. Grimaldi, A. Pescap, A. Salvi, and S. Santini, "A fuzzy approach based on heterogeneous metrics for scaling out public clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2117–2130, Aug 2017.

[44] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[45] S. Yuan, S. Das, R. Ramesh, and C. Qiao, "Service agreement trifecta: Backup resources, price and penalty in the availability-aware cloud," *Information Systems Research*, 2018.

[46] M. Ficco, C. Esposito, F. Palmieri, and A. Castiglione, "A coral-reefs and game theory-based approach for optimizing elastic cloud resource allocation," *Future Generation Computer Systems*, vol. 78, pp. 343 – 352, 2018.

[47] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Computing Surveys*, vol. 47, no. 2, pp. 33:1–33:36, Dec. 2014.

[48] X. Ruan and H. Chen, "Performance-to-power ratio aware virtual machine (vm) allocation in energy-efficient clouds," in *2015 IEEE International Conference on Cluster Computing*, Sept 2015, pp. 264–273.

[49] M. A. Khan, A. Paplinski, A. M. Khan, M. Murshed, and R. Buyya, *Dynamic Virtual Machine Consolidation Algorithms for Energy-Efficient Cloud Resource Management: A Review*. Springer International Publishing, 2018, pp. 135–165.

[50] Y. Liu, X. Ruan, S. Cai, R. Li, and H. He, "An optimized vm allocation strategy to make a secure and energy-efficient cloud against co-residence attack," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, March 2018, pp. 349–353.

[51] F. Teng, L. Yu, T. Li, D. Deng, and F. Magoulès, "Energy efficiency of vm consolidation in iaas clouds," *Journal of Supercomputing*, vol. 73, no. 2, pp. 782–809, Feb. 2017.

[52] D. Paul, W.-D. Zhong, and S. K. Bose, "Energy efficiency aware load distribution and electricity cost volatility control for cloud service providers," *Journal of Network and Computer Applications*, vol. 59, no. C, pp. 185–197, Jan. 2016.

[53] M. Xu, A. V. Dastjerdi, and R. Buyya, "Energy efficient scheduling of cloud application components with brownout," *IEEE Transactions on Sustainable Computing*, vol. 1, no. 2, pp. 40–53, 2016.

[54] G. Procaccianti, H. Fernández, and P. Lago, "Empirical evaluation of two best practices for energy-efficient software development," *Journal of Systems and Software*, vol. 117, no. C, pp. 185–198, Jul. 2016.

[55] X. Li, M. A. Salehi, M. Bayoumi, and R. Buyya, "Cvss: A cost-efficient and qos-aware video streaming using cloud services," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 106–115.

[56] S. G. Domanal and G. R. M. Reddy, "An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment," *Future Generation Computer Systems*, vol. 84, pp. 11 – 21, 2018.

[57] V. Singh, I. Gupta, and P. K. Jana, "A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources," *Future Gen. Computer Systems*, vol. 79, pp. 95–110, 2018.

[58] P. D. Kaur and I. Chana, "A resource elasticity framework for qos-aware execution of cloud applications," *Future Generation Computer Systems*, vol. 37, pp. 14 – 25, 2014.

[59] S. Singh and I. Chana, "Qos-aware autonomic resource management in cloud computing: A systematic review," *ACM Computing Surveys*, vol. 48, no. 3, pp. 42:1–42:46, Dec. 2015.

[60] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Proceedings of the 2012 International Conference on Computer Science and Electronics Engineering - Volume 01*, ser. ICCSEE '12, 2012, pp. 647–651.

[61] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 95–108, 2017.

[62] H. M. Demoulin, I. Pedisich, L. T. X. Phan, and B. T. Loo, "Automated detection and mitigation of application-level asymmetric dos attacks," in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, ser. SelfDN 2018. New York, NY, USA: ACM, 2018, pp. 36–42.

[63] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: A building block for automated diagnosis and

control," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, ser. OSDI'04. USENIX Association, 2004, pp. 16–16.

[64] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: A reinforcement learning approach to virtual machines auto-configuration," in *Proc. of the 6th Intl. Conf. on Autonomic Computing*, 2009, pp. 137–146.

[65] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, Jan. 2012.

[66] J. Rao, X. Bu, C. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," in *IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2011, pp. 45–54.

[67] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*. Springer, 2018, pp. 103–130.

[68] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 372–382.

[69] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 253–272, 2013.

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

[71] N. Mi, L. Cherkasova, K. Ozonat, J. Symons, and E. Smirni, "Analysis of application performance and its change via representative application signatures," in *Proc. of the 2008 IEEE Network Operations and Management Symposium*, April 2008, pp. 216–223.

[72] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Automated anomaly detection and performance modeling of enterprise applications," *ACM Transactions on Computer Systems*, vol. 27, no. 3, pp. 6:1–6:32, Nov. 2009.

[73] A. Brunnert and H. Krcmar, "Continuous performance evaluation and capacity planning using resource profiles for enterprise applications," *Journal of Systems and Software*, vol. 123, no. Supplement C, pp. 239 – 262, 2017.

[74] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds," in *Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, ser. DSN '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1–12.

[75] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, Dec. 2009.

[76] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 644–651.

[77] B. Hong, F. Peng, B. Deng, Y. Hu, and D. Wang, "Dac-hmm: Detecting anomaly in cloud systems with hidden markov models," *Concurrency and Computation:Practice and Experience*, vol. 27, no. 18, pp. 5749–5764, Dec. 2015.

[78] T. Chen, R. Bahsoon, and X. Yao, "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems," *ACM Computing Surveys*, vol. 51, no. 3, pp. 61:1–61:40, Jun. 2018.

[79] H. Hoffman, "Seec: A framework for self-aware management of goals and constraints in computing systems (power-aware computing, accuracy-aware computing, adaptive computing, autonomic computing)," Ph.D. dissertation, Cambridge, MA, USA, 2013.

[80] M. S. Aslanpour, M. Ghobaei-Arani, and A. N. Toosi, "Auto-scaling web applications in clouds: A cost-aware approach," *Journal of Network and Computer Applications*, vol. 95, pp. 26 – 41, 2017.

[81] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th International Conference on Autonomic Computing*, ser. ICAC '09. New York, NY, USA: ACM, 2009, pp. 117–126.

[82] A. Al-Shishtawy and V. Vlassov, "Elastman: Elasticity manager for elastic key-value stores in the cloud," in *Proceedings of the ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. ACM, 2013, pp. 7:1–7:10.

[83] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *IEEE Network Operations and Management Symposium - NOMS 2010*, April 2010, pp. 479–486.

[84] L. Lu, J. Yu, Y. Zhu, and M. Li, "A double auction mechanism to bridge users task requirements and providers resources in two-sided cloud markets," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 720–733, April 2018.

[85] W. Fang, X. Yao, X. Zhao, J. Yin, and N. Xiong, "A stochastic control approach to maximize profit on service provisioning for mobile cloudlet platforms," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 4, pp. 522–534, April 2018.

[86] D. Grimaldi, V. Persico, A. Pescape, A. Salvi, and S. Santini, "A feedback-control approach for resource management in public clouds," in *IEEE Global Communications Conference*, Dec 2015, pp. 1–7.

[87] S. Gong, B. Yin, W. Zhu, and K. Cai, "Adaptive resource allocation of multiple servers for service-based systems in cloud computing," in *IEEE 41st Annual Computer Software and Applications Conference*, vol. 2, 2017, pp. 603–608.

[88] P. Lama, Y. Guo, C. Jiang, and X. Zhou, "Autonomic performance and power con-

trol for co-located web applications in virtualized datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1289–1302, 2016.

[89] W. H. Kwon and S. H. Han, *Receding horizon control: model predictive control for state models*. Springer Science & Business Media, 2006.

[90] G. Mencagli, M. Vanneschi, and E. Vespa, "Reconfiguration stability of adaptive distributed parallel applications through a cooperative predictive control approach," in *Euro-Par 2013 Parallel Processing*, F. Wolf, B. Mohr, and D. an Mey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 329–340.

[91] D. Ardagna, M. Ciavotta, and R. Lancellotti, "A receding horizon approach for the runtime management of iaas cloud systems," in *Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Sept 2014, pp. 445–452.

[92] D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero, "A hierarchical receding horizon algorithm for qos-driven control of multi-iaas applications," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.

[93] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *IEEE 4th International Conference on Cloud Computing*, 2011, pp. 500–507.

[94] E. Incerto, M. Tribastone, and C. Trubiani, "Software performance self-adaptation through efficient model predictive control," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017, 2017, pp. 485–496.

[95] J. P. Buzen and A. W. Shum, "Masf - multivariate adaptive statistical filtering." in *Int. CMG Conference*. Computer Measurement Group, 1995, pp. 1–10.

[96] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, May 2011, pp. 385–392.

[97] O. Ibidunmoye, E. B. Lakew, and E. Elmroth, "A black-box approach for detecting systems anomalies in virtualized environments," in *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*, Sept 2017, pp. 22–33.

[98] M. Peiris, J. H. Hill, J. Thelin, S. Bykov, G. Kliot, and C. Konig, "Pad: Performance anomaly detection in multi-server distributed systems," in *Proceedings of the 7th IEEE Intl. Conf. on Cloud Computing*, June 2014, pp. 769–776.

[99] T. Matsuki and N. Matsuoka, "A resource contention analysis framework for diagnosis of application performance anomalies in consolidated cloud environments," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ser. ICPE '16. New York, NY, USA: ACM, 2016, pp. 173–184.

[100] M. Wajahat, A. Gandhi, A. Karve, and A. Kochut, "Using machine learning for black-box autoscaling," in *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, Nov 2016, pp. 1–8.

[101] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 32–32.

[102] J. Cao, B. Yu, F. Dong, X. Zhu, and S. Xu, "Entropy-based denial of service attack detection in cloud data center," in *Proceedings of the Second International Conference on Advanced Cloud and Big Data*, Nov 2014, pp. 201–207.

[103] H. Shi, H. Li, D. Zhang, C. Cheng, and W. Wu, "Efficient and robust feature extraction and selection for traffic classification," *Computer Networks*, vol. 119, pp. 1 – 16, 2017.

[104] S. Ajila and A. Bankole, "Using machine learning algorithms for cloud client prediction models in a web vm resource provisioning environment," *Transactions on Machine Learning and Artificial Intelligence*, vol. 4, no. 1, p. 28, 2016.

[105] X. Li, A. Ventresque, J. Iglesias, and J. Murphy, "Scalable correlation-aware virtual machine consolidation using two-phase clustering," in *Proceedings of the 2015 In-*

ternational Conference on High Performance Computing Simulation (HPCS), July 2015, pp. 237–245.

[106] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the 8th IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[107] ——, "Isolation-based anomaly detection," vol. 6, no. 1. ACM Transactions on Knowledge Discovery from Data, 2012, pp. 3:1–3:39.

[108] G. Pang, L. Cao, L. Chen, D. Lian, and H. Liu, "Sparse modeling-based sequential ensemble learning for effective outlier detection in high-dimensional numeric data," *Thirty-Second AAAI Conference on Artificial Intelligence.*, pp. 3892–3899, 2018.

[109] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 410–417.

[110] M. Duggan, J. Duggan, E. Howley, and E. Barrett, "A reinforcement learning approach for the scheduling of live migration from under utilised hosts," *Memetic Computing*, Dec 2016.

[111] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Piscataway, NJ, USA: IEEE Press, 2017, pp. 64–73.

[112] D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan, "Learn-as-you-go with megh: Efficient live migration of virtual machines," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2608–2609.

[113] K. Lolos, I. Konstantinou, V. Kantere, and N. Koziris, "Adaptive state space partitioning of markov decision processes for elastic resource management," in *IEEE 33rd International Conference on Data Engineering*. IEEE, 2017, pp. 191–194.

[114] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, Jun. 2011.

[115] M. Melo, J. Araujo, R. Matos, J. Menezes, and P. Maciel, "Comparative analysis of migration-based rejuvenation schedules on cloud availability," in *IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2013, pp. 4110–4115.

[116] M. A. Mukwevho and T. Celik, "Toward a smart cloud: A review of fault-tolerance methods in cloud systems," *IEEE Transactions on Services Computing*, 2018.

[117] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *Proceedings of the IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124–137, Apr. 2005.

[118] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen, "A cost-aware auto-scaling approach using the workload prediction in service clouds," *Information Systems Frontiers*, vol. 16, no. 1, pp. 7–18, Mar. 2014.

[119] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, ser. CLOUD '12. IEEE Computer Society, 2012, pp. 423–430.

[120] N. Grozev and R. Buyya, "Multi-cloud provisioning and load distribution for three-tier applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 3, pp. 13:1–13:21, Oct. 2014.

[121] G. Moltó, M. Caballer, and C. de Alfonso, "Automatic memory-based vertical elasticity and oversubscription on cloud platforms," *Future Generation Computer Systems*, vol. 56, no. C, pp. 1–10, Mar. 2016.

[122] M. Turowski and A. Lenk, *Vertical Scaling Capability of OpenStack*. Springer, 2015, pp. 351–362.

[123] Y. M. Teo and R. Ayani, "Comparison of load balancing strategies on cluster-based web servers," *SIMULATION*, vol. 77, no. 5-6, pp. 185–195, 2001.

[124] G. Soni and M. Kalra, "A novel approach for load balancing in cloud data center," in *Proceedings of the 2014 IEEE International Advance Computing Conference (IACC)*, Feb 2014, pp. 807–812.

[125] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003, pp. 309–320.

[126] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch, "Balancing load in stream processing with the cloud," in *2011 IEEE 27th International Conference on Data Engineering Workshops*, April 2011, pp. 16–21.

[127] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2007, pp. 17–17.

[128] M. Sommer, M. Klink, S. Tomforde, and J. Hhner, "Predictive load balancing in cloud computing environments based on ensemble forecasting," in *Proc. of the 2016 IEEE Intl. Conf. on Autonomic Computing*, July 2016, pp. 300–307.

[129] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, pp. 5:1–5:14.

[130] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 350–357, 2002.

[131] S. Agrawal and J. Agrawal, "Survey on anomaly detection using data mining techniques," *Procedia Computer Science*, vol. 60, pp. 708–713, 2015.

[132] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000, pp. 427–438.

[133] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996, pp. 226–231.

[134] Y. Zhu, K. M. Ting, and M. J. Carman, "Density-ratio based clustering for discovering clusters with varying densities," *Pattern Recogn.*, vol. 60, no. C, pp. 983–997, 2016.

[135] E. Stripling, B. Baesens, B. Chizi, and S. vanden Broucke, "Isolation-based conditional anomaly detection on mixed-attribute data to uncover workers' compensation fraud," *Decision Support Systems*, vol. 111, pp. 13 – 26, 2018.

[136] 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*, vol. 29, no. 18, p. e4169, 2017.

[137] T. Huang, Y. Zhu, Y. Wu, S. Bressan, and G. Dobbie, "Anomaly detection and identification scheme for vm live migration in cloud infrastructure," *Future Generation Computer Systems*, vol. 56, pp. 736–745, 2016.

[138] A. O. Ramirez, "Three-tier architecture," *Linux J.*, vol. 2000, no. 75es, Jul. 2000.

[139] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proceedings of the 1st Workshop on Cloud Computing and Its Applications*, vol. 8, Chicago, Illinois, USA, 2008.

[140] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "Stl: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, vol. 6, no. 1, pp. 3–73, 1990.

[141] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A novel technique for long-term anomaly detection in the cloud," in *Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing*, Philadelphia, PA, 2014, pp. 15–15.

[142] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240.

[143] X. Zhang, W. Dou, Q. He, R. Zhou, C. Leckie, R. Kotagiri, and Z. Salcic, "Lshiforest: A generic framework for fast tree isolation based ensemble anomaly analysis," San Diego, California, USA, 2017, pp. 983–994.

[144] J. Xiao, Z. Xiong, S. Wu, Y. Yi, H. Jin, and K. Hu, "Disk failure prediction in data centers via online learning," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. ACM, 2018, pp. 35:1–35:10.

[145] C. C. Aggarwal and P. S. Yu, "Outlier detection for high dimensional data," in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. ACM, 2001, pp. 37–46.

[146] M. H. Bhuyan, D. Bhattacharyya, and J. Kalita, "A multi-step outlier-based anomaly detection approach to network-wide traffic," *Information Sciences*, vol. 348, pp. 243 – 271, 2016.

[147] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387.

[148] H. Liu, X. Li, J. Li, and S. Zhang, "Efficient outlier detection for high-dimensional data," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017.

[149] F.-T. Liu, K.-M. Ting, and Z.-H. Zhou, "On detecting clustered anomalies using sciforest," in *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, 2010, pp. 274–290.

[150] F. Keller, E. Muller, and K. Bohm, "Hics: High contrast subspaces for density-based outlier ranking," in *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ser. ICDE '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1037–1048.

[151] H. Shi, H. Li, D. Zhang, C. Cheng, and X. Cao, "An efficient feature generation approach based on deep learning and feature selection techniques for traffic classification," *Computer Networks*, vol. 132, pp. 81 – 98, 2018.

[152] C. Pascoal, M. R. de Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, "Robust feature selection and robust pca for internet traffic anomaly detection," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 1755–1763.

[153] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*. London, UK, UK: Springer-Verlag, 2002, pp. 15–26.

[154] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 2017, pp. 90–98.

[155] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 94:1–94:45, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3136625

[156] D. Fesehaye, L. Singaraveluy, C. Chen, X. Huang, A. Banerjee, R. Zhou, and R. Somasundaran, "Group clustering using inter-group dissimilarities," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1011–1021.

[157] P. Filzmoser, R. Maronna, and M. Werner, "Outlier identification in high dimensions," *Comput. Stat. Data Anal.*, vol. 52, no. 3, pp. 1694–1711, Jan. 2008.

[158] T. de Vries, S. Chawla, and M. E. Houle, "Finding local anomalies in very high dimensional space," in *Proceedings of the 2010 IEEE International Conference on Data Mining*, 2010, pp. 128–137.

[159] A. Lazarevic and V. Kumar, "Feature bagging for outlier detection," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 2005, pp. 157–166.

[160] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[161] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[162] J. V. Bibal Benifa and D. Dejey, "Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Networks and Applications*, Jan 2018.

[163] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[164] N. Grozev and R. Buyya, "Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments," *The Computer Journal*, vol. 58, no. 1, pp. 1–22, 2013.

[165] RUBIS, "Rice university bidding system." [Online]. Available: http://rubis.ow2.org/

[166] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Generation Computer Systems*, vol. 78, pp. 191 – 210, 2018.

[167] K. Lolos, I. Konstantinou, V. Kantere, and N. Koziris, "Adaptive state space partitioning of markov decision processes for elastic resource management," in *IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017, pp. 191–194.

[168] M. Cheng, J. Li, and S. Nazarian, "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 129–134.

[169] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[170] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.